L'ÉLAGAGE DES RÉSEAUX NEURONAUX CONVOLUTIONNELS POUR FACILITER L'ANALYSE DU TRAFIC ROUTIER

par

Carl Lemaire

Mémoire présenté au Département d'informatique
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES

UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, 18 janvier 2019

Le 18 janvier 2019

le jury a accepté le mémoire de Monsieur Carl Lemaire dans sa version finale.

Membres du jury

Professeur Pierre-Marc Jodoin
Directeur de recherche
Département d'informatique

Jean Rouat
Professeur titulaire
Membre externe
Département de génie électrique et de génie informatique
Université de Sherbrooke

Professeur Manuel Lafond
Président-rapporteur
Département d'informatique

Sommaire

L'analyse automatique de vidéos est une direction prometteuse dans le domaine de l'analyse du trafic routier. Elle a de nombreux avantages par rapport aux méthodes antérieures (e.g. tubes pneumatiques), notamment un fonctionnement non altéré lors d'une accumulation de neige. Les réseaux de neurones à convolution, ou *convolutional neural networks* (CNN), sont tout indiqués pour localiser et classifier les véhicules dans des images routières.

Par le passé, les tâches spécifiques de localisation et classification de véhicules dans des images routières n'ont pas été des tâches usuelles dans le domaine d'étude des CNN. Une raison qui peut expliquer cette absence est le manque de données annotées publiquement accessibles. Pour surmonter ce problème, une base de données, nommée Mio-TCD (Miovision Traffic Camera Dataset), a été conçue et rendue publique par une équipe regroupant l'entreprise Miovision et les universités de Sherbrooke, Boston, Waterloo et Xiamen. De plus, une compétition internationale a été mise sur pied pour stimuler la recherche sur les CNN appliqués aux tâches sus-mentionnées. Cette compétition a permis de surpasser les méthodes de l'état de l'art en classification et en localisation. Le candidat a appliqué 6 modèles de CNN de l'état de l'art à Mio-TCD afin de fournir une base de comparaison.

Bien que les CNN aient prouvé leur efficacité pour les tâches d'inférence visuelle, leur usage dans des dispositifs à puissance limitée, tels que des caméras routières, est difficile dû à la consommation élevée de ressources généralement requise par les CNN. Pour surmonter ce problème, plusieurs options sont envisageables ; notamment l'optimisation du matériel et du lien entre le matériel et le logiciel, ou la conception manuelle d'architectures de CNN plus efficientes. Une direction particulièrement intéressante est *l'élagage de réseaux de neurones*, une famille de techniques permettant d'optimiser la structure d'un réseau neuronal, afin de réduire sa consommation de ressources, d'améliorer ses performances, ou

même les deux à la fois. Dans cet ouvrage, une nouvelle méthode est présentée ; cette méthode permet d'élaguer un CNN tout en respectant un budget de ressources. Aux niveaux d'élagage les plus sévères, cette méthode surpasse les méthodes de l'état de l'art.

Mots-clés: apprentissage machine ; apprentissage profond ; classification ; réseaux de neurones à convolution (CNN) ; élagage de réseaux de neurones (*pruning*).

Remerciements

Merci à mon superviseur, Pierre-Marc Jodoin, pour ses conseils pragmatiques et pour son dévouement exceptionnel.

Merci aux professeurs Manuel Lafond et Jean Rouat pour avoir gracieusement accepté d'évaluer ce mémoire.

Merci à Frédéric Branchaud-Charron pour sa générosité et son esprit d'équipe.

Merci à Charles Authier pour m'avoir toléré depuis le primaire, et pour sa bonne humeur contagieuse.

Merci à Sarah Leclerc pour son écoute et sa présence chaleureuse.

Merci aux gens du VITAL : Clément, Philippe, Faezeh, Antoine, Jon et Thierry pour tant de moments de découverte et de plaisir.

Merci à Antoine Boutin, mon grand ami, frère spirituel, grâce à qui la vie est une aventure toujours plus enrichissante.

Merci à ma famille : Olivette, Noémie, Jacques et Annie, pour leur support et leur amour inconditionnel.

Abréviations

CNN *Convolutional neural network*, ou réseau de neurones à convolution

OA Opération(s) arithmétique(s)

SGD *Stochastic gradient descent*, ou descente de gradient stochastique

MV Estimateur du maximum de vraisemblance

MAP *Maximum a posteriori*

Table des matières

Liste des figures

Introduction

L'analyse du trafic routier est essentielle pour le bon fonctionnement de nos infrastructures de transport, et conséquemment pour le maintien et le développement de notre société. Il existe plusieurs technologies permettant la collecte de données sur la circulation ; desquelles la collecte par caméra est souvent la plus avantageuse. L'efficacité de cette approche dépend de l'analyse d'images par réseaux de neurones. Or, bien que l'inférence à même la caméra soit souvent préférable, celle-ci demande des réseaux neuronaux exceptionnellement efficients.

0.1 La collecte de données routières essentielles

Les organisations responsables des infrastructures routières doivent assurer la collecte de données sur la circulation afin de maintenir ou améliorer la sécurité et la qualité de vie de la population [65]. Les données recueillies sont le dénombrement des véhicules de différentes catégories (camion, vélo, piéton, etc.), avec l'information temporelle et fréquentielle des passages. Les données de trafic routier informent les firmes d'ingénierie lors de la planification, construction ou réparation de routes, de pistes cyclables et de voies réservées, entre autres projets. Le bon fonctionnement des feux de circulation et des péages dépend de ces données. Elles permettent aussi de prédire les accidents potentiels et de les éviter en effectuant des changements dans la géométrie routière. Aussi, ces données aident aux décisions quant au climat et au transport actif. Il convient de bien choisir la méthode qui permettra d'effectuer les mesures, afin d'optimiser le temps et les dépenses requis pour un projet. Les technologies utilisées pour effectuer la cueillette se déclinent en deux catégories : les méthodes intrusives et les méthodes non-intrusives [81]. Parmi les méthodes non-intrusives, la collecte par caméra est particulièrement avantageuse, pour des raisons qui

seront décrites dans les paragraphes qui suivent. Seront aussi présentées des contributions au domaine auxquelles le candidat a participé.

0.1.1 Méthodes antérieures de cueillette

Les technologies intrusives sont installées sur le pavé ou sous celui-ci [81]. Par exemple, des tubes pneumatiques peuvent être installés sur la route. Cette solution est la moins coûteuse, mais n'est que temporaire : elle n'est ni durable ni compatible avec les opérations de déneigement. D'autres exemples de dispositifs sont les boucles à induction et les détecteurs piézoélectriques. Ces appareils nécessitent une excavation afin d'être installés. Ces dispositifs détectent les variations du champ électromagnétique causées par le passage des véhicules et la pression exercée par leur poids, respectivement. Étant sous terre, ces installations résistent aux intempéries. Cependant, elles sont coûteuses à installer et à réparer.

Pour éviter de placer un dispositif sur le pavé ou de creuser dans la route pour l'installer, il est possible de se tourner vers les méthodes non-intrusives, utilisant des appareils fonctionnant à partir du bord de la chaussée. Parmi celles-ci, il existe, entre autres, la mesure infrarouge et d'ondes radio. Bien qu'elles permettent de compter un nombre de véhicules, il est difficile d'en obtenir de l'information sur le type de véhicule. Or, l'utilisation de caméras combinées avec l'analyse d'images est une option qui procure de nombreux avantages. En plus d'offrir une installation non-intrusive, donc peu coûteuse, elle permet d'obtenir toutes les informations disponibles visuellement, telles que la catégorie du véhicule. Une seule caméra peut même couvrir plusieurs voies ou une intersection urbaine. Voilà qui explique la forte croissance de la demande pour des systèmes efficients d'analyse du trafic par caméra.

0.1.2 Cueillette par analyse d'images

L'extraction d'informations à partir d'un signal vidéo requiert l'usage de logiciels complexes. La difficulté du traitement d'images routières vient, entre autres, de la vaste variabilité de l'apparence des véhicules. Forme et couleur du véhicule, point de vue et éclairage de la scène, nombreux sont les facteurs qui influent sur l'image qui sera captée. Il n'est pas réaliste d'espérer obtenir des résultats de classification satisfaisants en spécifiant formellement une liste de caractéristiques à rechercher dans la matrice formée par l'image.

L'apprentissage automatique offre une combinaison de techniques et de modèles permettant de surmonter ces défis. Le modèle qui performe le mieux sur les images est le CNN (c.f. section 1.4.2).

L'usage de caméras présente un choix important à faire, soit celui de décider si l'analyse d'image sera faite par un système centralisé, ou assurée à même le dispositif de capture. L'approche centralisée peut à première vue sembler plus avantageuse, en évitant la conception d'un système embarqué, ou en permettant des itérations de développement plus courtes. Cependant, le débit d'information à transmettre à partir des caméras vers le serveur est substantiel et peut causer problème. En effet, l'usage d'appareils connectés à un réseau cellulaire est souvent l'option la plus pratique, et même parfois la seule option envisageable lorsqu'un lien internet filaire n'est pas disponible (e.g. autoroutes). Les tarifs élevés des réseaux cellulaires rendent prohibitif le coût de l'envoi d'un signal de haute qualité. Or, il est possible de réduire grandement la quantité d'information à transmettre, et ce en analysant les images à même le dispositif d'acquisition. Cela permet alors d'envoyer le dénombrement pour chaque catégorie de véhicules, au lieu d'envoyer des images à haute résolution et à haute fréquence (et de recevoir une faramineuse facture de données cellulaires). Le traitement d'images embarqué présente ses propres défis, dont l'utilisation de modèles lourds sur des plateformes aux ressources computationnelles limitées. Cette problématique est introduite à la section 0.2, et approfondie au chapitre 2.

0.1.3 Mio-TCD

Pour stimuler la recherche quant aux méthodes appliquant les CNN à des tâches d'analyse de trafic routier, la base de données Mio-TCD fut créée [1]. Elle contient 786 702 images annotées de véhicules circulant sur des réseaux routiers ; il s'agit, au moment de sa création, de la plus vaste base de données du genre [62]. Elle a été mise sur pied par *Miovision Technologies Inc.*, en collaboration avec l'Université de Sherbrooke, l'Université de Boston et l'Université de Xiamen. Afin d'encourager les chercheurs à travailler avec Mio-TCD, un événement appelé « Traffic Surveillance Workshop and Challenge » a été organisé en 2017, en marge de la prestigieuse conférence CVPR (*Computer Vision and Pattern Recognition*). Cette compétition consiste d'un volet classification et d'un volet localisation, correspon-

1. http://tcd.miovision.com

dant aux deux sous-ensembles de données. Afin de fournir un niveau de performance de référence, les méthodes de l'état de l'art ont été évaluées selon les critères de ladite compétition. Le candidat a participé à cet effort en implémentant et en entraînant six modèles de CNN sur la tâche de classification. Les modèles sont énumérés dans l'article journal décrivant Mio-TCD (c.f. annexe A).

0.2 L'élagage, pour des réseaux neuronaux efficients

Tel que mentionné précédemment, des applications telles que l'analyse du trafic routier peuvent bénéficier de réseaux neuronaux fonctionnant avec des ressources computationnelles restreintes. Plusieurs approches permettent d'obtenir des réseaux efficients (i.e. offrant une bonne performance par rapport aux ressources computationnelles requises) [40, 1, 48, 57, 58, 67, 71]. D'abord, des architectures neuronales ont été conçues manuellement afin d'être hautement efficientes. D'autre part, il existe des méthodes automatiques, dont certaines permettent *d'élaguer* des réseaux neuronaux, c'est-à-dire d'éliminer des neurones superflus dans les modèles.

0.2.1 Conception d'architectures neuronales

Il existe des heuristiques qui guident la conception des réseaux de neurones. L'une des plus populaires est celle qui suggère de doubler la *largeur* du réseau (i.e. le nombre de caractéristiques) à chaque fois qu'une « synthèse spatiale » (*pooling*) est effectuée. Cependant, la performance des architectures utilisant des *bottlenecks* suggère qu'un arrangement plus complexe quant à la largeur des couches peut permettre de réduire le fardeau computationnel d'un réseau, tout en conservant sa puissance [40]. Un *bottleneck* est une séquence de couches dont les largeurs internes sont moindres que celles des extrémités. Or, devant l'efficacité de l'apprentissage *end-to-end* [2], comment pourrait-on appliquer cette approche basée sur les données afin d'apprendre non seulement les poids, mais aussi l'architecture d'un réseau neuronal ?

2. L'apprentissage *end-to-end*, ou « de bout en bout », consiste à ne pas appliquer sur les données des méthodes d'extraction de caractéristiques conçues manuellement. Celles-ci sont remplacées par des couches neuronales entraînées sur les données.

0.2.2 Optimisation automatique d'architectures

L'optimisation d'architectures neuronales est un sujet de recherche en pleine effervescence. Une des approches consiste à rechercher dans un vaste espace d'architectures possibles en entraînant un agent à « comprendre » quelles caractéristiques des structures sont plus efficientes pour la tâche à résoudre [103, 54, 77, 76]. L'agent synthétise des architectures, reçoit une évaluation de leur performance, pour ainsi se raffiner et produire de meilleures architectures, et ainsi de suite. Une autre approche consiste à optimiser un réseau spécifique existant et performant, mais trop gourmand. Cette technique, appelée *élagage*, permet d'identifier des structures peu (ou pas) utiles, afin de les retirer ; potentiellement pour les remplacer par des structures plus efficientes [51, 32, 31, 58, 1, 26, 7]. Les méthodes d'élagage sont variées ; elles diffèrent, entre autres, par leur façon de gérer les interactions entre la structure du réseau et son contenu (i.e. ses poids). Elles diffèrent aussi par leur définition de l'objectif de réduction de la taille du réseau : certaines méthodes cherchent un compromis entre la taille et la performance [51, 32, 31, 58, 1] ; d'autres cherchent le meilleur réseau pour un budget de ressources donné [26, 7].

0.2.3 Élagage avec budget

Dans une situation où le matériel d'inférence impose des contraintes importantes (e.g. caméra avec analyse d'image embarquée), il peut être avantageux de fixer un budget à respecter pour les ressources utilisées par un réseau neuronal. Il importe alors de choisir quelle métrique sera contrainte par un budget. Le nombre total de poids uniques du réseau est une de ces métriques. Cette mesure est particulièrement indiquée pour réduire le temps de téléchargement d'un réseau [30]. Une autre mesure est le nombre total de neurones, laquelle est mieux corrélée que la précédente avec le fardeau computationnel du modèle [3]. Cependant, la métrique qui quantifie le mieux le fardeau computationnel est le nombre d'opérations arithmétiques (OA) exécutées lors de l'inférence (par définition). Malgré cela, le nombre de neurones peut être un substitut avantageux au nombre d'OA (c.f. chapitre 3). Pour imposer un budget, il importe aussi de choisir une méthode d'optimisation qui

3. Dans les couches à convolution, le nombre de neurones correspond au volume des *feature maps*, c'est-à-dire l'aire des *feature maps* multipliée par leur nombre, et ce pour chaque couche. En comparaison, le nombre de poids uniques ne tient pas compte de l'aire des *feature maps*.

permettra de converger vers une solution tout en respectant la contrainte. Ce choix est discuté au chapitre 3 ; alors qu'une nouvelle approche est proposée.

Avant d'approfondir l'élagage au chapitre 2, les fondements de cette approche, et plus généralement de l'apprentissage automatique, seront présentés au chapitre 1. Ensuite, au chapitre 3, le candidat présentera une nouvelle méthode d'élagage avec budget, appliquée notamment à la tâche de classification Mio-TCD.

Chapitre 1

Algorithmes d'apprentissage

Ce chapitre présente brièvement les notions de probabilités, de théorie de l'information et d'apprentissage automatique sur lesquelles se basent les chapitres suivants.

1.1 Probabilités

Une **variable aléatoire** est une variable dont la valeur est aléatoire. Une variable aléatoire est généralement notée par une lettre majuscule, alors qu'une réalisation qu'elle peut prendre est notée par une lettre minuscule. Par exemple, $P(X = x)$ indique la probabilité que la variable X prenne la valeur x. Toute variable aléatoire est définie par une distribution de probabilité, c'est-à-dire une fonction qui indique, dans le cas discret, la probabilité associée à chaque valeur possible. Dans le cas continu, la fonction indique la densité de probabilité associée à une valeur réelle du domaine de la variable.

Une **fonction de masse** $P(x)$ détermine la probabilité associée à chaque valeur entière x que peut prendre une variable aléatoire discrète. La somme des probabilités pour l'ensemble des valeurs possibles doit être égale à 1. Cette exigence s'applique également au cas continu, mais puisqu'il y a une infinité de valeurs possibles, la probabilité associée à chacune d'elles tend vers 0 ; c'est pourquoi il convient d'associer à chaque valeur une densité de probabilité, au lieu d'une masse. Une **densité de probabilité** $p(x)$ est l'équivalent, pour une variable continue, de la fonction de masse d'une variable discrète. L'intégrale de

la densité doit être égale à 1. Un fait important est que la densité peut prendre des valeurs supérieures à 1 ; par exemple, une fonction prenant la valeur 2 sur l'intervalle $[0, 0.5]$, et la valeur 0 ailleurs, est une densité valide, car l'aire sous la courbe est $0.5 \times 2 = 1$. La densité doit être non-négative.

L'**espérance** d'une fonction $f(x)$ par rapport à la distribution P est la valeur moyenne de la fonction si $X \sim P$, c.-à-d. X tirée de P. Lorsque x prend des valeurs discrètes, l'espérance est définie par la somme pondérée

$$\mathbb{E}_P[f(x)] = \sum_X P(x)f(x),$$

où $P(x)$ est la fonction de masse de P. Lorsque x prend des valeurs continues, l'espérance est définie par l'intégrale

$$\mathbb{E}_p[f(x)] = \int_X p(x)f(x)dx,$$

où $p(x)$ est la densité de probabilité de $X \sim p$.

1.2 Théorie de l'information

La théorie de l'information a été développée pour améliorer l'efficience des transmissions de messages par ondes radio [86]. Son objectif est d'envoyer un maximum d'informations avec un minimum de symboles, à travers un canal bruité, tout en assurant l'intégrité des messages. Les notions d'information et d'entropie issues de ce domaine sont très utiles dans le cadre de l'apprentissage automatique.

L'**information** $I_P(x)$ exprime la « surprise » encourue lorsque l'événement $X = x$ se produit, avec $X \sim P$. Lorsqu'un événement se produit avec probabilité 1, la surprise est nulle, alors que si sa probabilité tend vers zéro, la surprise tend vers l'infini. Cette idée est incarnée par l'équation suivante :

$$I_P(x) = -\log P(x).$$

Si le logarithme en base 2 est utilisé, $I_P(x)$ s'exprime en **bits** ; dans ce contexte, un bit est la quantité d'information reçue en observant un événement de probabilité $1/2$. En effet, $-\log_2 1/2 = 1$.

L'**entropie** $H(P)$ est l'espérance de l'information $I_P(x)$, ou le nombre de bits d'information moyen obtenu en observant $X \sim P$:

$$H(P) = \mathbb{E}_P[-\log P(x)] = -\sum_X P(x) \log P(x).$$

Dans le cadre de l'apprentissage automatique, le besoin de comparer deux distributions de probabilité est fréquent. Pour y répondre, l'**entropie relative** est utilisée. Cette mesure est souvent appelée **divergence de Kullback-Leibler**, ou simplement *KL divergence*. Elle est définie comme suit :

$$D_{\mathrm{KL}}(P||Q) = -\mathbb{E}_P[\log Q(x) - \log P(x)] = \sum_X P(x) \log \frac{P(x)}{Q(x)}.$$

Cette mesure s'interprète comme la quantité d'information supplémentaire nécessaire pour envoyer un message contenant des symboles tirés de P, en utilisant un encodage conçu pour minimiser la longueur des messages tirés de la distribution Q [1]. Plus intuitivement, en apprentissage automatique, l'entropie relative est vue comme une distance entre deux distributions. Il est important de mentionner que cette mesure n'est pas symétrique : $D_{\mathrm{KL}}(P||Q) \neq D_{\mathrm{KL}}(Q||P)$. L'ordre de P et Q dépend du problème. D_{KL} sert souvent à approximer $P(x)$ avec $Q(x)$: si il faut s'assurer que $Q(x)$ soit grand lorsque $P(x)$ est grand, $D_{\mathrm{KL}}(P||Q)$ est préférable ; à l'inverse, s'il faut s'assurer que $Q(x)$ soit petit lorsque $P(x)$ est petit, $D_{\mathrm{KL}}(Q||P)$ est préférable. Le lecteur intéressé peut se référer à la figure 3.6 de Goodfellow *et al.* [25].

La minimisation de l'entropie relative $D_{\mathrm{KL}}(P||Q) = \mathbb{E}_P[\log P(x)] - \mathbb{E}_P[\log Q(x)]$ par rapport à $Q(x)$ est équivalente à minimiser l'expression $-\mathbb{E}_P[\log Q(x)]$, car le terme de gauche ne dépend pas de $Q(x)$. Cette expression s'appelle l'**entropie croisée** :

$$H(P, Q) = -\mathbb{E}_P[\log Q(x)],$$

une mesure fréquemment utilisée comme fonction de coût, notamment pour les problèmes de classification. Dans la présente section, toutes les variables aléatoires sont discrètes ; cependant, toutes les formules sont valides pour des variables continues, en remplaçant la

1. Traduction libre de la définition de Goodfellow *et al.* [25].

fonction de masse par la densité, et la somme par l'intégrale.

1.3 Apprentissage automatique

Il convient de donner une définition de l'apprentissage dans le contexte de l'apprentissage automatique. Selon Mitchell [66] : « [traduction libre] un logiciel apprend au sujet de la tâche T par l'expérience E, si sa performance à la tâche T, telle que mesurée par M, s'améliore avec l'expérience E. » Cette phrase tisse des liens serrés entre trois notions fondamentales (T, M, E) qui seront définies dans ce qui suit.

1.3.1 Tâche, performance et données

La **tâche** T est l'action que le logiciel effectue sur un **exemple** donné. Il est important de ne pas confondre tâche et processus d'apprentissage : la tâche est le sujet de l'apprentissage ; et non l'apprentissage lui-même. La tâche qui sera étudiée dans cet ouvrage est la **classification**. Un algorithme de classification reçoit un exemple et doit donner la catégorie à laquelle l'exemple appartient ; pour la classification de véhicules, l'algorithme recevra une image d'une voiture, et devra la classifier dans une catégorie (parmi un ensemble de k catégories). À chaque catégorie est associé un indice ; c'est celui-ci que l'algorithme retournera. Formellement, un modèle de classification est une fonction $f : \mathbb{R}^n \to \{1, \ldots, k\}$. Une autre tâche est la **régression**. Cette tâche consiste à inférer un scalaire à partir d'un exemple. Formellement, la fonction apprise est $f : \mathbb{R}^n \to \mathbb{R}$. Une tâche qui a un vecteur en sortie au lieu d'un scalaire peut être formulée soit comme plusieurs régressions (si les éléments à inférer sont indépendants) ou comme une « prédiction structurée » (si les éléments sont interdépendants) [25]. Les autres tâches abordées par l'apprentissage automatique incluent la transcription, la traduction, la synthèse et l'estimation de densité de probabilité, pour ne donner que quelques exemples.

La **mesure de performance** M sert à évaluer la compétence du modèle appris. Pour des tâches avec des sorties discrètes, l'*accuracy* est utilisée (ce terme est rarement traduit, mais l'Office québécois de la langue française suggère le terme « exactitude »). L'*accuracy* est la proportion d'exemples bien classés. Pour des tâches à sortie continue, comme la régression, l'erreur quadratique moyenne est souvent le premier choix. Lorsque le processus

d'apprentissage optimise une densité de probabilité, l'entropie croisée est souvent utilisée (plus de détails à la section 1.3.2). Bien qu'il soit possible de mesurer la performance de l'algorithme quant aux exemples fournis pendant l'entraînement, il est davantage intéressant de mesurer la compétence du modèle sur de nouvelles données : la **généralisation**. Pour ce faire, une manière simple est de séparer les exemples en un ensemble d'entraînement et un **ensemble de test** ; ce dernier sera caché lors de l'apprentissage, mais c'est lui qui sera utilisé pour mesurer la performance de l'algorithme.

L'expérience E fait référence à la source des exemples donnés au processus d'apprentissage. Bien que l'expérience soit souvent contenue entièrement dans un **ensemble de données**, ce n'est pas toujours le cas. Par exemple, l'apprentissage par renforcement est une technique d'apprentissage qui expérimente avec un environnement : les actions prises par l'agent influencent ses observations, et *vice-versa* ; l'expérience n'est pas décrite par un ensemble de données fixe. Dans cet ouvrage, l'expérience correspondra toujours à un ensemble de données ; le terme expérience sera donc mis de côté. En classification et en régression, l'ensemble de données associe un exemple avec une **cible** ; par exemple, une image de véhicule avec l'indice de sa catégorie. Pour d'autres tâches, comme l'estimation de densité, seulement les exemples sont fournis ; ces tâches sont considérées comme étant apprises de manière non-supervisée.

Maintenant que les notions de tâche, de mesure de performance et d'ensemble de données ont été introduites, le processus d'apprentissage sera formulé en un problème d'optimisation.

1.3.2 Estimateur du maximum de vraisemblance

Soit une distribution $p_{\text{data}}(x)$ à approximer, modélisée par $p_{\text{model}}(x \mid \theta) = p_{\text{data}}(x)$, et $\hat{\theta} = g(\mathbb{X})$ une estimation de θ, où $\mathbb{X}$ est un ensemble de données tiré de $p_{\text{data}}(x)$. En apprentissage automatique, le choix le plus commun pour l'estimateur g est l'**estimateur**

du maximum de vraisemblance, défini par :

$$\begin{aligned}\hat{\theta}_{\text{MV}} &= \arg\max_{\theta} p_{\text{model}}(\mathbb{X} \mid \theta) \\ &= \arg\max_{\theta} \prod_{\boldsymbol{x}\in\mathbb{X}} p_{\text{model}}(\boldsymbol{x} \mid \theta) \\ &= \arg\max_{\theta} \sum_{\boldsymbol{x}\in\mathbb{X}} \log p_{\text{model}}(\boldsymbol{x} \mid \theta) \\ &= \arg\max_{\theta} \mathbb{E}_{X\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\boldsymbol{x} \mid \theta)],\end{aligned} \tag{1.1}$$

où $\boldsymbol{x}$ est un vecteur et $\hat{p}_{\text{data}}$ est la distribution empirique de $\mathbb{X}$. À noter que la dernière ligne de l'équation 1.1 est équivalente à la minimisation de l'entropie croisée $H(\hat{p}_{\text{data}}, p_{\text{model}})$. La mise en correspondance de p_{model} et de $\hat{p}_{\text{data}}$ peut être formulée comme la minimisation d'une entropie relative (*KL divergence*), mais tel que vu dans la section 1.2, cette minimisation est équivalente à celle de l'entropie croisée.

Il convient de noter une autre formulation du problème. L'estimateur du **maximum *a posteriori*** (MAP) est similaire à celui du maximum de vraisemblance. L'avantage du MAP est qu'il permet d'optimiser la vraisemblance de θ par rapport à $\mathbb{X}$, tout en indiquant une distribution *a priori* sur θ. Au lieu de maximiser $\log p_{\text{model}}(\boldsymbol{x} \mid \theta)$, ce sera :

$$\log p_{\text{model}}(\theta \mid \boldsymbol{x}) \propto \log p_{\text{model}}(\boldsymbol{x} \mid \theta) + \log p(\theta) \tag{1.2}$$

qui sera maximisé. L'approche MAP consiste donc à ajouter un terme $\log p(\theta)$ à la fonction optimisée. Par exemple, si une distribution gaussienne est supposée pour θ, le terme à ajouter sera proportionnel au carré de la somme des éléments de θ [25]. L'*a priori* gaussien est très commun ; il est souvent appelé **régularisation L2** (ou L_2). Pour alléger les formules, le reste de cette section sera basée sur le maximum de vraisemblance, sans terme *a priori*.

La définition de l'équation 1.1, comme formulée, s'applique directement à la tâche d'estimation de densité de probabilité. Cependant, cette définition se transpose facilement à des tâches d'inférence, telles que la classification, comme suit :

$$\begin{aligned}\hat{\theta}_{\text{MV}} &= \arg\max_{\theta} p_{\text{model}}(\mathbf{Y} \mid \mathbf{X}, \theta) \\ &= \arg\max_{\theta} \mathbb{E}_{X,Y\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(y \mid \boldsymbol{x}, \theta)],\end{aligned} \tag{1.3}$$

où $\mathbf{X}$ et $\mathbf{Y}$ sont les exemples et les cibles correspondantes, respectivement. Pour obtenir un classifieur à partir d'un modèle $P_{\text{model}}(y \mid \boldsymbol{x}, \theta)$, il suffit d'appliquer un $\arg\max$ à sa sortie :

$$y_{\text{pred}} = \arg\max_{y} P_{\text{model}}(y \mid \boldsymbol{x}, \theta), \tag{1.4}$$

où y_{pred} est l'indice de la catégorie inférée par le modèle. Le modèle de classification le plus simple se base sur la régression linéaire, et se nomme **régression logistique** multinomiale :

$$\begin{aligned} P_{\text{model}}(y \mid \boldsymbol{x}, \theta) &= \text{Softmax}(\Theta \boldsymbol{x}) \\ \text{Softmax}(\boldsymbol{h}) &= \frac{[e^{h_1}, \ldots, e^{h_n}]}{\sum_i e^{h_i}}, \end{aligned} \tag{1.5}$$

où $\boldsymbol{x}, \boldsymbol{h} \in \mathbb{R}^n$. $\Theta \in \mathbb{R}^{m \times n}$ constitue les paramètres à optimiser. La fonction Softmax peut être vue comme une généralisation de la sigmoïde à $n > 2$ classes (pour la classification binaire, $P(Y = 0) = 1 - P(Y = 1)$, alors seulement $P(Y = 1)$ est inféré, avec une régression linéaire et l'application subséquente de la sigmoïde).

Maintenant que le problème d'optimisation permettant d'entraîner un modèle a été formulé, le choix de l'algorithme d'optimisation sera discuté. La construction de modèles plus complexes sera abordée plus loin.

1.3.3 Descente de gradient stochastique

Tout problème d'apprentissage automatique requiert l'optimisation d'une **fonction objectif** : par exemple, la minimisation d'une fonction de coût (*loss*), ou la maximisation d'une fonction qui estime la somme cumulative des récompenses (en apprentissage par renforcement). Dans la suite de cet ouvrage, les problèmes d'optimisation seront formulés comme des problèmes de minimisation (la maximisation est possible en minimisant l'opposé). Lorsque la fonction objectif n'est pas convexe, comme c'est généralement le cas en apprentissage profond (c.f. prochaine section), l'optimum ne peut pas être calculé en forme close. Ainsi, une méthode itérative devra être utilisée pour trouver le minimum global, ou un minimum local satisfaisant.

Soit une fonction de coût $f : \mathbb{R}^n \to \mathbb{R}$ à minimiser. Le gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ est un vecteur qui contient chacune des dérivées partielles de f. La direction du gradient est celle pour laquelle l'accroissement de la fonction est maximale. Ainsi, pour minimiser f, il faut se

déplacer dans la direction opposée, en effectuant une **descente de gradient** :

$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x}),$$

où le pas $\epsilon \in \mathbb{R}$ est choisi pour que f soit approximativement linéaire sur $\boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. Dans le cadre de l'entraînement d'un classifieur, la fonction objectif à minimiser est la suivante (c.f. équation 1.3) :

$$\begin{aligned} J(\theta) &= \mathbb{E}_{X,Y \sim \hat{p}_{\text{data}}}[L(\boldsymbol{x}, y, \theta)] \\ L(\boldsymbol{x}, y, \theta) &= -\log p_{\text{model}}(y \mid \boldsymbol{x}, \theta). \end{aligned}$$

Le gradient de cette fonction est calculé comme suit :

$$\nabla_\theta J(\theta) = \mathbb{E}_{X,Y \sim \hat{p}_{\text{data}}}[\nabla_\theta L(\boldsymbol{x}, y, \theta)].$$

Cette opération de complexité $O(|\mathbf{X}|\ |\theta|)$ doit être calculée pour chacune des nombreuses itérations de descente de gradient, ce qui rend prohibitif le coût computationnel de cette méthode avec de grands ensembles de données. La **descente de gradient stochastique** (SGD) se base sur la reformulation de l'espérance en une composition d'espérances :

$$\nabla_\theta J(\theta) = \mathbb{E}_{X,Y \sim \hat{p}_{\text{data}}}[\nabla_\theta L(\boldsymbol{x}, y, \theta)] = \mathbb{E}_{\mathbb{B} \sim \hat{p}_{\text{data}}}\Big[\mathbb{E}_{X,Y \sim \mathbb{B}}[\nabla_\theta L(\boldsymbol{x}, y, \theta)]\Big],$$

où $\mathbb{B}$ est une ***mini-batch***, soit un sous-ensemble des données échantillonné aléatoirement. En estimant le gradient avec une nouvelle *mini-batch* $\mathbb{B}$ à chaque itération de SGD, la direction prise sera bonne (en espérance). De plus, la taille de $\mathbb{B}$ est petite et restera fixe même si l'ensemble de données s'agrandit ; permettant ainsi un temps de calcul raisonnable pour chaque itération.

Maintenant que la théorie sur laquelle se base l'apprentissage automatique a été présentée, les modèles complexes qu'elle permet d'entraîner seront abordés. Dans la section suivante, l'accent sera mis sur les réseaux neuronaux.

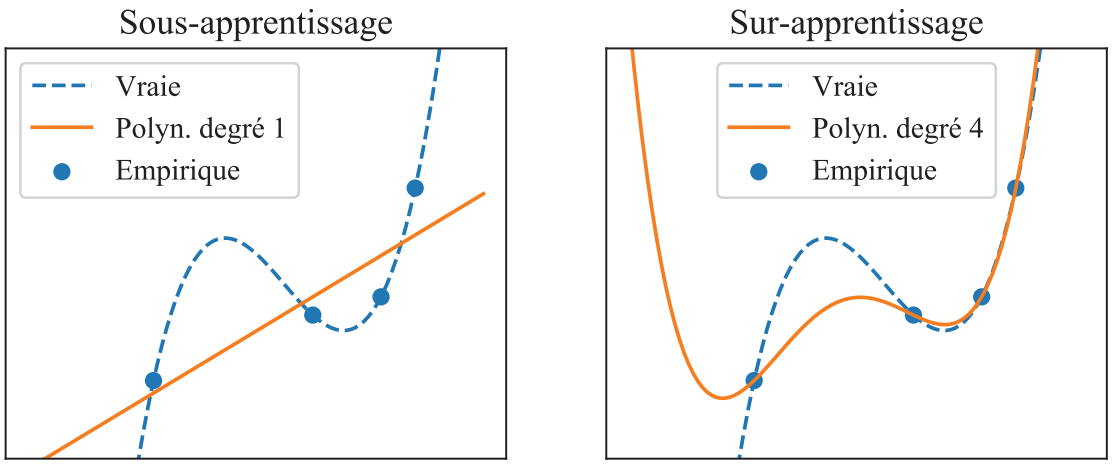


figure 1.1 – **Sous-apprentissage et sur-apprentissage.** Dans cette illustration, les données empiriques proviennent d'une polynomiale de degré 3 (en pointillé). *(Gauche)* Tenter d'approximer la vraie fonction avec une droite résulte en un sous-apprentissage : l'approximation n'est pas fidèle, car trop grossière. *(Droite)* Le choix d'une polynomiale de degré 4 résulte en un sur-apprentissage : bien qu'elle explique parfaitement toutes les données empiriques, elle correspond à une hypothèse trop compliquée, qui ne généralise pas à tout le domaine.

1.4 Modèles

Il convient de trouver un modèle ayant une puissance adaptée à l'ensemble de données et à la tâche à accomplir. Le **sous-apprentissage** se manifeste lors de l'utilisation d'un modèle trop peu puissant, alors que le **sur-apprentissage** s'observe lorsqu'un modèle utilise trop de degrés de liberté. Les figures 1.1 et 1.2 illustrent ces phénomènes.

Par exemple, l'utilisation d'une régression logistique multinomiale (c.f. équation 1.5) ne permet pas à elle seule d'atteindre une performance satisfaisante pour la tâche de classification de véhicules routiers. Le sous-apprentissage apparent dans cette situation peut être expliqué par plusieurs facteurs. D'abord, le modèle manque de capacité de représentation. La **compositionnalité**, telle qu'offerte par les réseaux neuronaux multicouches, permet de surmonter ce problème en décomposant la tâche en une séquence à différents niveaux d'abstraction (c.f. section 1.4.1). Or, un autre problème est que le modèle doit transformer les images en vecteurs pour constituer les données à fournir à la régression logistique ; cette transformation détruit les relations spatiales des pixels. Une solution possible

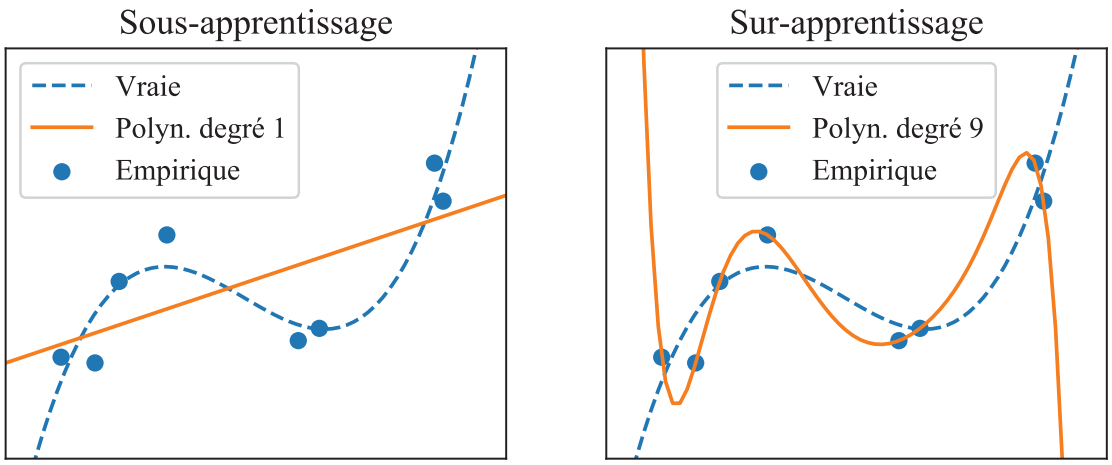


figure 1.2 – **Sous- et sur-apprentissage avec observations bruitées.** (Complément à la figure 1.1.) Lorsque les données sont bruitées, ce qui est généralement le cas, le sur-apprentissage se manifeste par une trop grande sensibilité au bruit.

à ce problème est d'effectuer une **extraction de caractéristiques** grâce à des algorithmes comme HOG (*Histograms of Oriented Gradients* [13]) ou SIFT (*Scale-Invariant Feature Transform* [59]), qui produisent des vecteurs tenant compte des structures spatiales dans les images. Cependant, la méthode la plus efficace est l'usage d'un réseau neuronal convolutionel (CNN), qui apprend automatiquement les meilleures caractéristiques à extraire pour chaque ensemble de données ; ce type de modèle est abordé à la section 1.4.2.

1.4.1 Perceptron multicouche (MLP)

Le *multi-layer perceptron* (MLP) est un type de **réseau neuronal multicouche**, qui bâtit sur le Perceptron (un modèle probabiliste simple du neurone biologique présenté en 1958 [84]). Le MLP est basé sur l'idée d'une séquence de transformations linéaires (type de transformation utilisé par la régression linéaire multinomiale) :

$$\hat{\boldsymbol{y}} = W_n \dots W_2 W_1 \boldsymbol{x}, \tag{1.6}$$

où W_i sont des matrices de poids, et $\boldsymbol{x}$ est le vecteur en entrée, et $\hat{\boldsymbol{y}}$ est le vecteur en sortie. Dans le modèle défini dans l'équation précédente, puisque les transformations sont li-

néaires, elles peuvent être simplifiées en une seule : $W_n \dots W_2 W_1 \boldsymbol{x} = W' \boldsymbol{x}$. Pour permettre au réseau d'apprendre par niveaux d'abstraction via la compositionalité, il faut rendre les transformations non-linéaires. Pour ce faire, l'usage d'une **fonction d'activation** (aussi appelée non-linéarité) est nécessaire. Les valeurs résultantes d'une transformation linéaire sont passées élément-par-élément dans une fonction telle que la sigmoïde, la tangente hyperbolique, ou « ReLU » (*rectified linear unit* : $\max(0, x)$), comme suit :

$$\hat{\boldsymbol{y}} = \sigma\Big(W_n \dots \sigma\Big(W_2\ \sigma(W_1 \boldsymbol{x})\Big)\Big), \tag{1.7}$$

où σ est une fonction d'activation. Les éléments des vecteurs $\boldsymbol{x}$ et $\hat{\boldsymbol{y}}$, et ceux des vecteurs intermédiaires résultants d'une transformation, sont appelés **neurones**. Les éléments des matrices W_i sont appelés **poids**, car chacun d'eux correspond au poids d'une connexion entre deux neurones. En effet, dans une couche de transformation linéaire, un neurone est le résultat de la somme des neurones de la couche précédente, pondérée par un vecteur de poids. Le terme **apprentissage profond** désigne l'usage de réseaux neuronaux multicouches dans le cadre de l'apprentissage automatique.

1.4.2 Réseau neuronal convolutionnel (CNN)

Le *convolutional neural network* (CNN) est un modèle similaire au MLP ; cependant, il utilise un type de couche spécial appelé **couche à convolution**. Une convolution 2D est une opération dans laquelle un **filtre** est glissé sur une image pour calculer le degré de correspondance du motif à chaque position dans l'image. Le filtre a une petite taille, généralement 3×3, mais parfois un peu plus ; il comporte un nombre de canaux égal à celui des images qui seront analysées. Le résultat de la convolution d'un filtre avec une image se nomme carte d'activation, ou ***feature map***. Soit une image I représentée par un tenseur d'ordre 3 indicé par i, j et $c \in \{r, g, b\}$, un filtre W de taille $3 \times 3 \times 3$ indicé par $i' \in \{-1, 0, 1\}, j' \in \{-1, 0, 1\}$ et c ; la valeur d'un pixel (i, j) dans la *feature map* O est calculée comme suit :

$$O[i, j] = \sum_{c, i', j'} W[c, i', j'] \cdot I[c, i + i', j + j']. \tag{1.8}$$

Une distinction importante entre la transformation linéaire et la convolution est que dans la première, chaque poids est associé à un seul neurone, tandis que dans la seconde, chaque poids intervient pour chaque neurone de la *feature map* ; les poids sont **partagés**. À noter aussi que la formule précédente permet de calculer une seule *feature map*, mais que les couches à convolution comportent généralement plusieurs filtres, donc plusieurs *feature maps* en sortie. La sortie d'une couche à convolution est donc un tenseur d'ordre 3 contenant une pile de *feature maps*.

La dernière couche d'un CNN de classification est souvent une régression logistique multinomiale (c.f. équation 1.5) ; or, cette opération ne peut pas recevoir en entrée une pile de *feature maps*. Cette couche doit recevoir un vecteur qui décrit globalement l'image en entrée du réseau. C'est pourquoi l'information est résumée sur les axes spatiaux par des synthèses spatiales qui sont effectuées à différentes profondeurs du CNN. Généralement, ces opérations sont effectuées dans des couches dédiées, appelées ***pooling** layers* (« couches de mise en commun »). Le cas le plus fréquent est le *max-pooling* 2×2, qui divise les *feature maps* en cases de taille 2×2, et ne garde que la valeur maximale de chaque case ; cette opération réduit les dimensions d'un facteur 2. Après plusieurs *poolings*, la pile des *feature maps* a une taille 1×1 ; cette pile sera interprétée comme un vecteur qui décrit globalement l'image d'entrée. Ce vecteur, parfois appelé **feature vector**, peut être utilisé pour la classification, ou pour toute autre tâche nécessitant une représentation vectorielle d'une image. Par exemple, il est possible de donner ce vecteur en entrée à un réseau neuronal récurrent pour générer une phrase décrivant l'image [93].

Bref, dans le présent chapitre, les fondements de probabilités et de théorie de l'information derrière l'apprentissage automatique ont été présentés. De plus, le problème d'optimisation qu'est l'apprentissage a été formulé, et une description des modèles permettant l'inférence a été faite ; certains modèles étant simples, et d'autres plus complexes, tels que les réseaux neuronaux profonds et les CNN. Dans le chapitre suivant, l'élagage des réseaux neuronaux sera abordé.

Chapitre 2

Élagage de réseaux neuronaux

Plusieurs approches très différentes permettent de réduire le fardeau computationnel des réseaux neuronaux. Dans ce chapitre, l'accent sera mis sur les méthodes d'élagage, mais une vue d'ensemble de la variété des méthodes sera abordée dans la première section. Les sections subséquentes traiteront de l'apprentissage de parcimonie (i.e. une méthode d'élagage), et de l'élagage avec budget.

2.1 Réduction de la surparamétrisation

Une hypothèse populaire dans le domaine des réseaux neuronaux veut que les réseaux soient surparamétrisés, c'est-à-dire qu'ils font usage d'un plus grand nombre de degrés de liberté que nécessaire. Par exemple, Frankle *et al.* supportent empiriquement l'hypothèse selon laquelle la valeur d'initialisation des poids donne souvent lieu à des neurones inutiles (jusqu'à 90% des neurones) [20]. D'autre part, Denil *et al.* ont pu, dans les meilleurs cas, prédire 95% des poids à partir des 5% restants [15]. En pratique, lors de la conception d'architectures neuronales profondes, lorsque le nombre de paramètres est augmenté, les modèles obtenus sont plus performants. Le surapprentissage est évité grâce à des méthodes de régularisation efficaces, et la stabilité assurée par des méthodes de normalisation éprouvées (c.f. chapitre 7 et section 8.7.1 de [25]). Cette observation a causé une tendance à l'augmentation de la taille des réseaux, avec même certains travaux, tels que

« BigGAN » [5], qui ont pour principale contribution l'utilisation de réseaux significativement plus gros qu'auparavant. Bien que des réseaux énormes peuvent être utilisés lorsque les ressources computationnelles disponibles sont quasi-illimitées, comme dans un cadre compétitif ou dans un contexte d'infonuagique, il est irresponsable d'ignorer complètement l'efficience des modèles (ne serait-ce que d'un point de vue écoénergétique). En fait, l'efficience est indispensable pour une myriade d'applications ; notamment les applications sur plateformes mobiles ou embarquées. Le présent ouvrage utilise l'élagage pour permettre l'analyse d'images à même une caméra routière ; cependant, il existe d'autres approches. Sera abordée d'abord la compression, qui est une méthode qui permet de compresser les paramètres du réseau, c'est-à-dire de trouver une représentation des poids qui utilise moins de bits. Sera décrite ensuite l'élagage, qui cherche à éliminer des neurones complètement. Finalement, sera expliquée la *Knowledge Distillation* (« distillation de connaissances »), permettant de distiller les connaissances d'un gros réseau dans un réseau plus petit.

2.1.1 Compression

Le terme « compression » a ici le même sens que dans « compression d'image ». Le but premier de la compression n'est pas de réduire la taille du réseau, mais plutôt de trouver une représentation efficiente de son « savoir », afin d'utiliser moins de stockage et de le télécharger rapidement. Certains travaux [16, 42] utilisent la décomposition en valeurs singulières afin de factoriser les matrices de poids du réseau en facteurs de rang moindre. Gong *et al.* [24] utilisent la quantification vectorielle afin de regrouper les filtres similaires dans des partitionnements pour ainsi ne garder qu'un représentant par groupe. La quantification scalaire peut aussi être utilisée afin de représenter chaque poids avec moins de bits. Bien que les méthodes de compression puissent parfois être adaptées à l'objectif de réduire le fardeau computationnel des réseaux, elles ne sont généralement pas conçues dans ce but.

2.1.2 Élagage

En sélectionnant des groupes de neurones à éliminer complètement, l'élagage s'attaque directement au fardeau computationnel d'un réseau. Puisque, dans un CNN, la vaste majorité de ce fardeau provient des couches de convolution, l'attention sera portée vers ces opérations uniquement. La sortie d'une convolution est un tenseur 3D contenant une pile

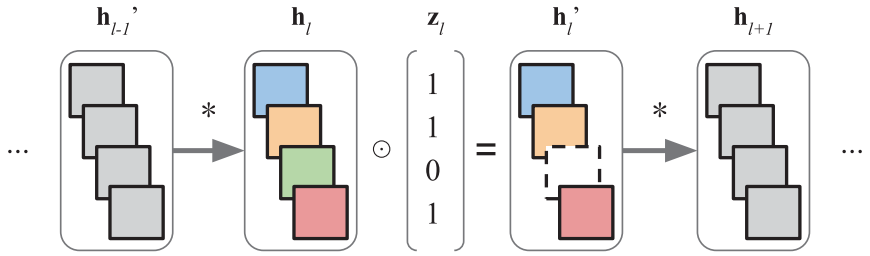


figure 2.1 – **Parcimonie au niveau des *feature maps*.** Dans ce schéma, $*$ représente la convolution, $\mathbf{h}_l$ représente les *feature maps* en sortie d'une couche de convolution, $\mathbf{z}_l$ est le masque de la couche l, et $\mathbf{h}'_l$ représente $\mathbf{h}_l$ après application du masque. Une *feature map* pointillée est remplie de zéros, et équivalente à une absence de *feature map*.

de *feature maps*. Bien qu'il soit possible d'élaguer selon une variété de granularités différentes, un groupement par *feature map* est particulièrement adapté dans un contexte de calcul parallèle. Cela est équivalent à chercher une parcimonie au niveau des filtres. Une telle parcimonie est illustrée à la figure 2.1. À chaque *feature map* est associée une « porte », soit une variable binaire qui détermine si le groupe de neurones doit être conservé ou non. Une valeur 1 pour cette variable indique que le groupe est conservé (n'est pas élagué) ; alors qu'une valeur 0 indique que le groupe n'est pas conservé (est élagué). L'ensemble des portes d'une couche sera nommé « masque de parcimonie » et noté $\mathbf{z}_l$, où l identifie la couche. Afin d'évaluer la performance du réseau élagué, les masques seront multipliés à la sortie des opérations de convolution. Lorsque la liste des filtres à éliminer sera confirmée, la structure du réseau pourra être modifiée en conséquence, et les masques pourront être jetés ; c'est alors que les gains en efficience seront observés.

Il existe plusieurs façons de déterminer la liste des filtres à éliminer. Certaines méthodes [51, 32] ont par le passé utilisé la dérivée seconde de la *loss* par rapport à chaque poids pour déterminer la sensibilité du réseau par rapport à ce poids ; afin d'ensuite éliminer les neurones n'affectant que peu (ou pas) la performance. Puisque le calcul de la dérivée seconde est très lourd avec les réseaux d'aujourd'hui, cette méthode a été mise de côté. Une approche [31] simple, légère et efficace consiste à utiliser la valeur absolue d'un poids pour estimer son importance. La sélection des filtres peut être faite selon un seuil, ou par tri, en prenant les N neurones (ou filtres) les moins importants selon le critère.

Bien que les masques de parcimonie puissent être déterminés à l'extérieur de la boucle

d'apprentissage grâce à des techniques telles que mentionnées ci-haut, les résultats empiriques suggèrent que les méthodes d'apprentissage de parcimonie offrent de meilleurs résultats (c.f. chapitre 3). Cette approche est abordée à la section 2.2.

2.1.3 *Knowledge Distillation*

Proposée par Hinton *et al.* [37], la « distillation de connaissances » est une approche complètement différente pour diminuer la taille d'un réseau. Elle requiert d'avoir en main un réseau performant mais coûteux. Ce réseau sera appelé « l'enseignant », et permettra d'entraîner un plus petit réseau, nommé « l'élève ». Dans la présente description, la technique sera appliquée à la tâche de classification. Alors que l'enseignant a été entraîné uniquement avec des vecteurs *one-hot* comme cible (indiquant la vraie classe de chaque échantillon), l'élève devra imiter les probabilités assignées à chaque classe par l'enseignant. Par exemple, au début de l'entraînement d'un modèle de classification de véhicules, il est très probable que le modèle assigne la classe « voiture » à un véhicule de la classe « pickup ». Étant données les similitudes entre ces deux classes (e.g. en comparaison avec la classe « bicyclette »), il n'est pas surprenant que les deux classes aient un léger chevauchement ; cela se traduit d'ailleurs dans les probabilités inférées par l'enseignant. Lors de l'entraînement de l'enseignant, ce genre de méprise (voiture et pickup) est autant pénalisé que la méprise d'un pickup pour une bicyclette. En fournissant les « connaissances inter-classes » de l'enseignant vers l'élève, l'entraînement de ce dernier est plus facile et requiert un réseau moins profond et moins large [8, 78]. En pratique, l'élève est entraîné avec une fonction objectif qui est une interpolation entre l'entropie croisée « vérité-terrain » $L_{\mathrm{CE}}(P_s, Y_{gt})$ et l'entropie croisée « enseignant » $L_{\mathrm{CE}}(P_s, P_t)$:

$$\begin{aligned} L_{\mathrm{KD}} &= (1-\alpha)L_{\mathrm{CE}}(P_s, Y_{gt}) + \alpha T^2 L_{\mathrm{CE}}(P_s, P_t) \\ P_t &= e^{Y_t/T} / \sum e^{Y_t/T}, \end{aligned} \tag{2.1}$$

où Y_t sont les scores inférés par l'enseignant et T est la température. P_s sont les probabilités inférées par l'élève, calculées de la même façon que P_t avec $T = 1$.

La *Knowledge Distillation* est orthogonale aux méthodes susmentionnées de réduction de la surparamétrisation ; elle peut même être combinée avec l'élagage (c.f. chapitre 3). La section suivante décrit l'approche d'élagage sur laquelle le candidat a concentré son étude.

2.2 Apprentissage de parcimonie

Il existe plusieurs méthodes permettant d'apprendre les masques $\mathbf{z}_l$ d'un réseau. Une des approches consiste à associer à chaque porte un paramètre continu à optimiser, où une valeur zéro pour ce paramètre équivaut à une porte fermée. Une autre approche consiste quant à elle à traiter les portes comme des variables aléatoires binaires. Dans ce dernier cas, c'est la paramétrisation de la distribution de chaque variable aléatoire qui sera optimisée.

2.2.1 Apprentissage de parcimonie direct

Il est possible de traiter chaque porte comme un paramètre à optimiser par SGD. C'est d'ailleurs l'approche utilisée par MorphNet [26]. Les gradients rétropropagés jusqu'à ces paramètres sont influencés par les deux termes de la fonction objectif : le terme d'attache aux données, et le terme de parcimonie. Bien que la mesure de parcimonie exacte soit la norme L_0, celle-ci n'est pas dérivable ; c'est pourquoi la norme L_1, étant dérivable, est souvent utilisée comme terme de parcimonie. Cette norme s'intègre à la fonction objectif complète comme suit, où L_D est le terme d'attache au données (e.g. entropie croisée) :

$$\begin{aligned} L(W, \mathbf{z}) &= L_D(W, \mathbf{z}) + \lambda\, L_{\text{parcim}}(\mathbf{z}) \\ L_{\text{parcim}}(\mathbf{z}) &= \sum_l ||\mathbf{z}_l||_1 = \sum_l \sum_i |z_{l,i}|. \end{aligned} \tag{2.2}$$

Le terme d'attache aux données L_D aura tendance a faire augmenter les $|z_{l,i}|$, alors que le terme de parcimonie L_{parcim} aura l'effet opposé. Dans la formulation actuelle, il est attendu que les *feature maps* non importantes se retrouveront avec $|z_{l,i}| = 0$; cependant, l'équilibre des forces des deux termes d'objectif pourrait être atteint sur n'importe quelle valeur. En particulier, des valeurs très proches de zéro minimisent le terme de parcimonie, mais ne permettent pas d'évaluer l'absence d'une *feature map* ; par exemple, un $z_{l,i} = 10^{-6}$ pourrait être contre-balancé par un poids d'intensité 10^6 dans une couche subséquente. Voilà un inconvénient qui n'existe pas avec l'apprentissage de parcimonie par *Dropout* (c.f. section 2.2.3). Avant d'introduire cette approche, il convient d'introduire les variables aléatoires « concrètes ».

2.2.2 Variables aléatoires concrètes

L'apprentissage de parcimonie par *Dropout* se base sur l'idée de traiter les portes comme des variables aléatoires de distribution Bernoulli, qui devraient être zéro le plus souvent possible pour que le réseau soit parcimonieux. Cependant, il n'est pas possible d'optimiser dans un réseau neuronal la paramétrisation d'une distribution discrète. L'échantillonnage à partir d'une distribution discrète fait intervenir l'opération $\arg\max$, et les gradients rétro-propagés par cette fonction ne peuvent qu'être nuls ou indéfinis. Pour résoudre ce problème, il est possible d'utiliser une relaxation continue de variables aléatoires discrètes.

Les variables aléatoires « concrètes » portent dans leur publication originale [64] le nom « concrete », qui signifie *CONcrete relaxation of disCRETE random variables*. Pour représenter une valeur discrète dans un réseau neuronal, il convient d'utiliser une représentation *one-hot*, c'est-à-dire une valeur sur le simplexe $\Delta_D^{n-1} = \{x \in \mathbb{N}^n \mid x_k \in \{0,1\},\ \sum_k x_k = 1\}$, où n est le nombre de valeurs différentes que la variable peut prendre. Cependant, une variable concrète, étant relaxée par rapport à sa contrepartie discrète, prend une valeur sur le simplexe $\Delta_C^{n-1} = \{x \in \mathbb{R}^n \mid x_k \in [0,1],\ \sum_k x_k = 1\}$. La distribution prend les paramètres $\alpha_k \in (0,\infty)$ et $\beta \in (0,\infty)$, où $\alpha_k / \sum_i \alpha_i$ donne la probabilité de l'événement k, et β est le degré de relaxation. Lorsque $\beta \to 0$, la distribution devient discrète (i.e. sur le simplexe Δ_D^{n-1}). La densité de la distribution est donnée par l'équation suivante, où α représente le vecteur des α_k :

$$p_{\alpha,\beta}(x) = (n-1)!\ \beta^{n-1} \prod_{k=1}^{n} \left(\frac{\alpha_k x_k^{-\beta-1}}{\sum_{i=1}^{n} \alpha_i x_i^{-\beta}} \right), \tag{2.3}$$

et la façon de générer un échantillon est la suivante, où Gumbel fait référence à la loi de Gumbel [29] :

$$\begin{aligned} X &\overset{d}{=} \mathrm{Softmax}((\log\alpha + G)/\beta) \\ G &\overset{d}{=} -\log(-\log U) \sim \mathrm{Gumbel}(0,1) \\ U &= [U_1, U_2, \ldots, U_n] \sim \mathcal{U}(0,1) \\ \mathrm{Softmax}(y) &= \frac{[e^{y_1}, e^{y_2}, \ldots, e^{y_n}]}{\sum_i e^{y_i}}. \end{aligned} \tag{2.4}$$

Les variables aléatoires **concrètes binaires** sont un cas particulier des variables concrètes

dans lequel $n = 2$, ce qui réduit le simplexe à un segment unidimensionnel (i.e. l'intervalle $[0, 1]$). La densité de cette distribution est la suivante, où α est un scalaire :

$$p_{\alpha,\beta}(x) = \frac{\beta\alpha x^{-\beta-1}(1-x)^{-\beta-1}}{(\alpha x^{-\beta} + (1-x)^{-\beta})^2}, \tag{2.5}$$

et sa fonction d'échantillonnage est la suivante, où $\mathrm{Logistic}$ fait référence à la loi logistique :

$$\begin{aligned} X &\overset{d}{=} \mathrm{Sigmoid}((\log\alpha + L)/\beta) \\ L &\overset{d}{=} \log U - \log(1-U) \sim \mathrm{Logistic}(0,1) \\ U &\sim \mathcal{U}(0,1) \\ \mathrm{Sigmoid}(y) &= (1+e^{-y})^{-1}. \end{aligned} \tag{2.6}$$

À noter que, dans la figure 2.2 (en haut à gauche), la distribution concrète binaire comporte, comme la distribution de Bernoulli, deux points où la densité tend vers l'infini. Au lieu d'ajuster $P(X = 1)$ comme pour Bernoulli, le paramètre de la concrète binaire ajuste $P(X > 0.5)$. À noter aussi que pour $\beta \to 0$, $P(X = 0) = 1 - P(X = 1)$.

Maintenant que la distribution concrète binaire a été introduite, il sera question de la distribution **concrète dure**. Il est de mise de motiver cette nouvelle distribution. Il faut comprendre que lors de l'apprentissage des masques $\mathbf{z}_l$, l'utilisation d'une valeur très proche de zéro pour une porte est très différente de l'utilisation de la valeur exactement zéro. Un signal très proche de zéro peut contenir de l'information utile (le signal peut être mis à l'échelle par une couche subséquente) ; voilà pourquoi, lorsque l'on veut tester l'importance d'un neurone, il faut multiplier sa sortie par exactement zéro. La valeur zéro étant celle à laquelle une porte doit arriver pour considérer un groupe de neurone comme élagué, il faut que cette valeur ait une masse substantielle dans la distribution. Cette exigence n'est pas satisfaite par la distribution concrète binaire (car $P(X = x) \to 0$ dans toute distribution strictement continue).

La distribution concrète dure, modification de la concrète binaire, est une distribution mixte (à la fois discrète et continue). Elle assigne une masse significative à chacun des événements $P(X = 0)$ et $P(X = 1)$, alors que les autres réalisations $X \in (0, 1)$ ont une densité de probabilité. Cette distribution mixte peut être vue comme une distribution composée ; pour en produire un échantillon, une distribution discrète avec trois événements

$X = 0$, $X = 1$ et $X \in (0, 1)$ peut être utilisée d'abord, avec une distribution continue ensuite, lorsque l'événement $X \in (0, 1)$ survient. Bien que la distribution soit composée, elle peut être décrite par une seule fonction de répartition (définie par morceaux). L'inverse de cette dernière, qui permet de produire des échantillons, est strictement continue. Une comparaison des distributions concrète binaire et concrète dure se trouve à la figure 2.2. La fonction d'échantillonnage de la concrète dure est la suivante :

$$\begin{aligned} X_{\text{dure}} &\stackrel{d}{=} \text{Clamp}_{0,1}\left[(\zeta - \gamma)X_{\text{cb}} + \gamma\right] \\ \text{Clamp}_{0,1}[y] &= \mathbb{1}(0 \leq y \leq 1)y + \mathbb{1}(y > 1), \end{aligned} \tag{2.7}$$

où X_{cb} est un échantillon de concrète binaire produit par l'équation 2.6, $\zeta > 1$ et $\gamma < 0$ décrivent un étirement centré en 0.5 appliqué à l'échantillon X_{cb}. L'étirement est mieux compris visuellement (c.f. figure 2.2).

2.2.3 Apprentissage de parcimonie par *Dropout*

L'apprentissage de parcimonie par *Dropout* est implémentée de manière similaire à la technique *Dropout* [69] servant à régulariser les réseaux neuronaux. Cette dernière consiste à multiplier les neurones en sortie d'une couche par une variable aléatoire de distribution Bernouilli pendant l'entraînement. Dans le cas qui nous occupe, au lieu d'utiliser une distribution Bernoulli avec un paramètre partagé pour la couche, c'est la distribution concrète dure qui sera utilisée, avec un paramètre α pour chaque *feature map*. Lors de l'exécution du graphe de calcul, les masques de parcimonie sont appliqués comme suit (c.f. figure 2.1) :

$$\mathbf{h}_{l+1} = f_{l+1}(\mathbf{h}_l \odot \mathbf{z}_l), \tag{2.8}$$

où f_l est la convolution à la couche l, et $\mathbf{z}_l$ est composé de variables aléatoires concrètes dures. Les $\mathbf{z}_l$ ne sont pas des paramètres à optimiser, mais bien des variables aléatoires. Il faut donc, à chaque itération d'entraînement, échantillonner une nouvelle valeur pour chaque élément de $\mathbf{z}_l$, de la façon suivante :

$$\mathbf{h}_{l+1} = f_{l+1}(\mathbf{h}_l \odot g(\boldsymbol{\alpha}_l, \mathbf{U}, \beta, \gamma, \zeta)), \tag{2.9}$$

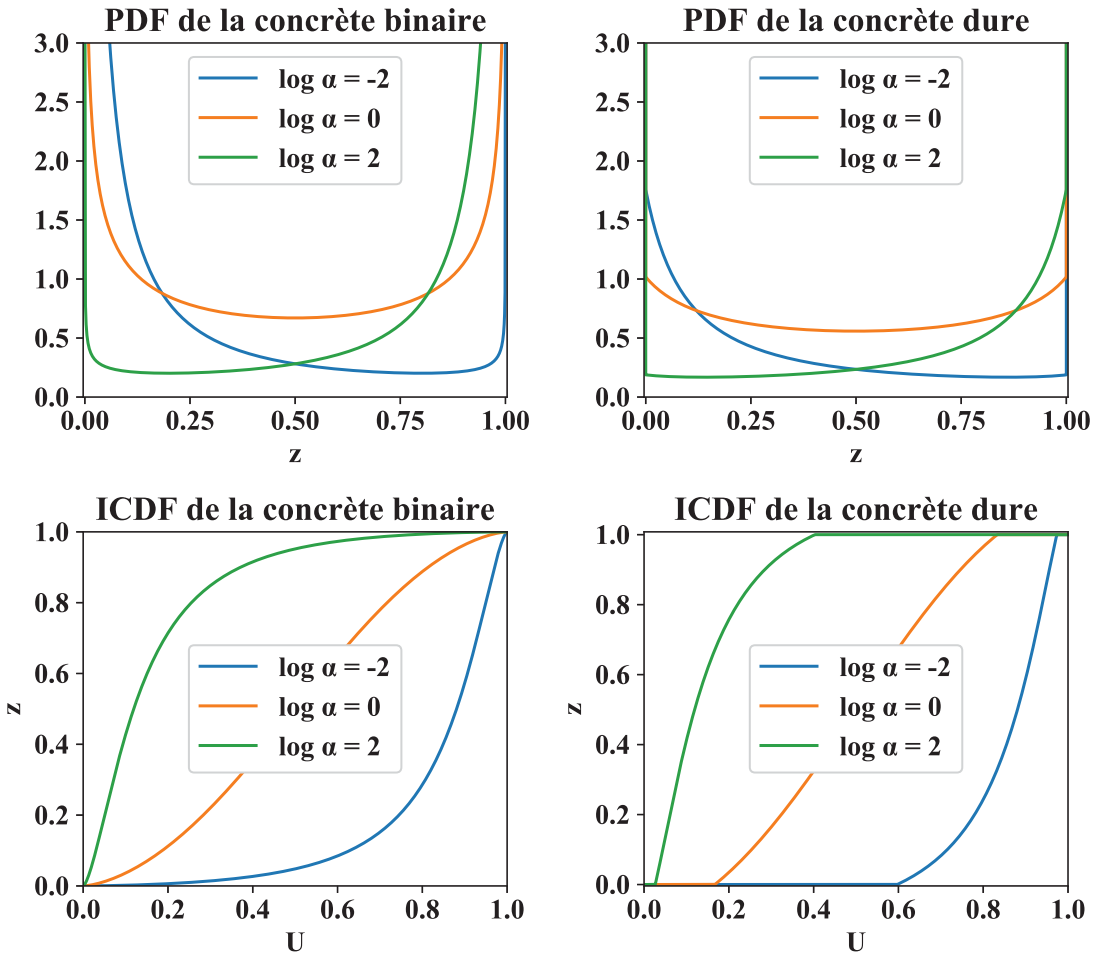


figure 2.2 – **Comparaison des distributions concrète binaire et concrète dure.** PDF signifie *probability density function* (i.e. la densité de probabilité), et ICDF signifie *inverse cumulative distribution function* (i.e. inverse de la fonction de répartition). Aux paramètres β, γ, ζ sont assignées les valeurs $2/3, -0.1, 1.1$.

où $g(\cdot)$ est la fonction d'échantillonage de l'équation 2.7, $\boldsymbol{\alpha}_l$ sont les paramètres à optimiser (un pour chaque porte), $U_i \sim \mathcal{U}(0,1)$, et β, γ, ζ sont fixés.

Les α seront optimisés selon une fonction objectif contenant un terme d'entropie croisée (comme pour tous les autres poids du réseau) ; cependant, ces paramètres seront aussi affectés par un terme de parcimonie. Ce terme est souvent formulé dans la littérature comme une entropie relative entre la distribution empirique des portes et une distribution *a priori* favorisant la parcimonie [48, 67]. En général, il faut minimiser l'espérance de la norme L_0 des portes. Lorsque la distribution concrète dure est utilisée, le terme de parcimonie minimise $P(z > 0 \mid \alpha)$, qui a une forme close :

$$L_{\text{parcim}}(\boldsymbol{\alpha}) = \sum_{\alpha \in \boldsymbol{\alpha}} P(z > 0 \mid \alpha) = \sum_{\alpha \in \boldsymbol{\alpha}} \text{Sigmoid}(\log \alpha - \beta \log \frac{-\gamma}{\zeta}). \tag{2.10}$$

La minimisation de ce terme contre le sur-apprentissage en réduisant le nombre de neurones dans le réseau (c.f. chapitre 3). Ainsi, il est possible d'utiliser ce terme de régularisation à la place de la norme L_2 (c.f. section 1.3.2).

2.3 Élagage avec budget

Maintenant que la technique d'apprentissage de parcimonie a été décrite, elle sera appliquée à l'élagage avec budget. Soit B le budget alloué pour le nombre de neurones total dans le réseau ; un problème de minimisation avec contrainte est formulé comme suit :

$$\begin{aligned} \min_{W, \boldsymbol{\alpha}} \quad & L_D(W, \boldsymbol{\alpha}) \\ \text{sujet à} \quad & ||\mathbf{z}||_0 \leq B, \end{aligned} \tag{2.11}$$

où L_D est la *loss* d'attache aux données. Il existe plusieurs moyens d'optimiser avec SGD en tenant compte de contraintes ; notamment, l'algorithme du lagrangien augmenté (c.f. annexe E de [2]), ainsi que l'algorithme des directions alternées (ADMM [3]), qui est basé sur le premier. Un autre exemple est l'utilisation d'une fonction barrière, une approche qui sera décrite ici.

Une fonction barrière tend vers l'infini au bord intérieur de la région acceptable d'une contrainte. Cette fonction $f(\cdot)$ peut être intégrée à la fonction objectif, ce qui permet de

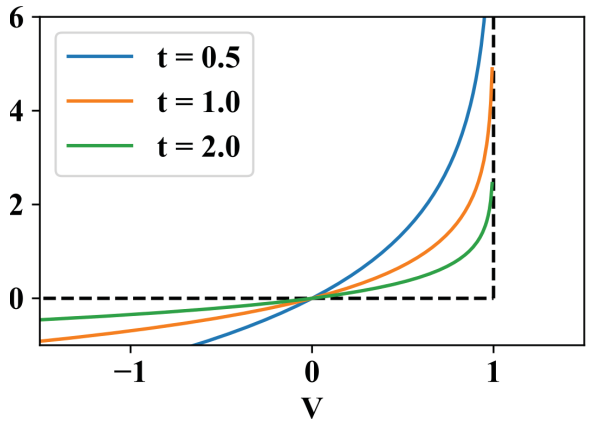


figure 2.3 – **Fonction barrière logarithmique.** La fonction barrière logarithmique, ou *logarithmic barrier function*, est définie par $f(x, y, t) = -(1/t)\log(y - x)$.

reformuler le problème en un problème de minimisation sans contraintes :

$$\min_{W,\boldsymbol{\alpha}} L_D(W, \boldsymbol{\alpha}) + f(||\mathbf{z}||_0, B), \tag{2.12}$$

où $\lim_{x \to y} f(x, y) = \infty$. Un exemple de fonction barrière est $f(x, y) = -\log(y - x)$. Puisque cette fonction sert à approximer une contrainte dure, un paramètre de relaxation t est utilisé pour améliorer l'approximation au long de l'optimisation : $f(x, y, t) = -(1/t)\log(y - x)$. Cette fonction est tracée à la figure 2.3. Deux inconvénients sont à noter : d'abord, $||\mathbf{z}||_0$ n'est pas dérivable et ne peut donc pas être utilisé dans la fonction objectif. De plus, au début de l'optimisation, $||\mathbf{z}||_0 > B$ nécessairement, car le réseau doit être élagué ; ce qui implique que la fonction barrière prend une valeur infinie. Une solution à ces problèmes est proposée au chapitre 3 sous la forme d'une nouvelle fonction barrière. Cette solution est aussi comparée à d'autres travaux d'élagage avec budget (c.f. section 3.2).

Chapter 3

Structured Pruning of Neural Networks with Budget-Aware Regularization

Cet article présente une nouvelle méthode d'élagage de réseaux de neurones. Cette méthode, nommée *Budget-Aware Regularization* (BAR), permet de spécifier un budget en termes d'un nombre maximal de neurones à utiliser dans les couches de convolution. La méthode offre à l'utilisateur un contrôle intéressant quant à la quantité de ressources utilisées par le CNN. De plus, BAR utilise l'apprentissage de parcimonie par *Dropout*, combinées à la *Knowledge Distillation*, pour optimiser la structure du réseau. En comparaison, les méthodes d'élagages précédentes ne permettent pas de spécifier directement un budget, et utilisent des techniques d'optimisation de la parcimonie moins performantes. La méthode a été évaluée sur les ensembles de données CIFAR10, CIFAR100 et Mio-TCD.

La technique BAR a été conçue conjointement par le candidat et par Pierre-Marc Jodoin ; il en est de même pour la rédaction de l'article. L'implémentation de la nouvelle méthode ainsi que des 7 méthodes de la littérature a été effectuée par le candidat ; tout comme pour la mise en place du protocole d'expérimentation, l'exécution des expériences et le dessin des figures. Andrew Achkar, scientifique des données sénior chez Miovision Technologies Inc. (l'entreprise qui a fourni les images de la base de données Mio-TCD) a été d'une aide précieuse en fournissant du matériel et nombre d'idées et de pistes de réflexions quant aux tâches de classification de véhicules dans des images routières.

Contributions

Les contributions au domaine présentées dans l'article sont listées ci-dessous.

— Présentation d'une nouvelle fonction objectif qui inclut une variante de la fonction barrière logarithmique, permettant d'entraîner et d'élaguer simultanément un CNN tout en respectant un budget de neurones ;

— Présentation d'une nouvelle variante de la méthode d'optimisation avec contrainte avec fonction barrière ;

— Démonstration de l'efficacité de la combinaison de l'apprentissage de parcimonie et *Knowledge Distillation* ;

— Confirmation de l'existence de la propriété de détermination automatique de la profondeur d'un réseau neuronal résiduel (une architecture neuronale performante et facile à entraîner [35]) ;

— Présentation du « bloc à connectivité mixte » (*mixed-connectivity block*), permettant de profiter d'une efficience maximale lors de l'élagage de réseaux neuronaux résiduels. Alors qu'un bloc résiduel ordinaire additionne toujours une précédente *feature map* avec une nouvelle, le *mixed-connectivity block* permet pour chaque *feature map* de soit garder la précédente *feature map* intacte, soit introduire une nouvelle là ou il y avait absence, ou encore effectuer une addition. Le concept est mieux expliqué grâce aux figures présentes dans l'article.

L'article a été soumis le 16 novembre 2018 en vue de la *Conference on Computer Vision and Pattern Recognition* (CVPR) 2019.

Structured Pruning of Neural Networks with Budget-Aware Regularization

Carl Lemaire[1], Andrew Achkar[2] and Pierre-Marc Jodoin[1]

[1]*Université de Sherbrooke, Sherbrooke, Canada*
[2]*Miovision Technologies Inc., Kitchener, Canada*

Abstract

Pruning methods have shown to be effective at reducing the size of deep neural networks while keeping accuracy almost intact. Among the most effective methods are those that prune a network while training it with a sparsity prior loss and learnable dropout parameters. A shortcoming of these approaches however is that neither the size nor the inference speed of the pruned network can be controlled directly; yet this is a key feature for targeting deployment of CNNs on low-power hardware. To overcome this, we introduce a budgeted regularized pruning framework for deep convolutional neural networks. Our approach naturally fits into traditional neural network training as it consists of a learnable masking layer, a novel budget-aware objective function, and the use of knowledge distillation. We also provide insights on how to prune a residual network and how this can lead to new architectures. Experimental results reveal that CNNs pruned with our method are more accurate and less compute-hungry than state-of-the-art methods. Also, our approach is more effective at preventing accuracy collapse in case of severe pruning; this allows us to attain pruning factors up to $16\times$ without significantly affecting the accuracy.

3.1 Introduction

Convolutional Neural Networks (CNN) have proven to be effective feature extractors for many computer vision tasks [34, 39, 50, 83]. The design of several CNN models involve many heuristics, such as using increasing powers of two as the number of feature maps, or *width*, of each layer. While such heuristics allow to achieve excellent results, they may be too crude in situations where the amount of compute power and memory is restricted, such as with mobile platforms. Thus arises the problem of finding the right number of layers that

solve a given task while respecting a budget. Since the number of layers depends highly on the effectiveness of the learned filters (and their combination), one cannot determine these hyper-parameters *a priori*.

Convolution operations constitute the main computational burden of a CNN. The execution of these operations benefit from a high degree of parallelism, which requires them to have regular structures. This implies that one cannot remove isolated neurons from a CNN filter as they must be full grids. To achieve the same effect as removing a neuron, one can *zero-out* its weights. While doing this reduces the theoretical size of the model, it does not reduce the computational demands of the model nor the amount of feature map memory. Therefore, to accelerate a CNN and reduce its memory footprint, one has to rely on *structured* sparsity pruning methods that aim at reducing the number of feature maps and not just individual neurons.

By removing unimportant filters from a network and retraining it, one can shrink a network while maintaining satisfying performance [31, 51]. This can be explained by the following hypothesis: the initial value of a filter's weights is not guaranteed to allow the learning of a useful feature; thus, a trained network might contain many expendable features [20].

Among the structured pruning methods, those that implement a sparsity learning (SL) framework have shown to be effective as pruning and training are done simultaneously [1, 48, 57, 58, 67, 71]. Unfortunately, most SL methods cannot prune a network while respecting a *neuron budget* imposed by the very nature of a device on which the network shall be deployed. As of today, pruning a network while respecting a budget can only be done by trial-and-error, typically by training multiple times a network with various pruning hyperparameters.

In this paper, we present a SL framework which allows for simultaneously learning and selecting filters of a CNN while respecting a neuron budget. Our main contributions are as follows:

— We present a novel objective function which includes a variant of the log-barrier [4] function for simultaneously training and pruning a CNN while respecting a total neuron budget;
— We propose a variant of the barrier method [4] for optimizing a CNN;
— We demonstrate the effectiveness of combining SL and knowledge distillation [37];

— We empirically confirm the existence of the *automatic depth determination* property of residual networks pruned with filter-wise methods, and give insights on how to ensure the viability of the pruned network by preventing "fatal pruning";
— We propose a new *mixed-connectivity block* which roughly doubles the effective pruning factors attainable with our method.

3.2 Previous Works

Compressing neural networks without affecting too much their accuracy implies that networks are often over-parametrized. Denil *et al.* [15] have shown that typical neural networks are over-parametrized; in the best case of their experiments, they could predict 95% of the network weights from the others. Recent work by Frankle *et al.* [20] support the hypothesis that a large proportion (typically 90%) of weights in standard neural networks are initialized to a value that will lead to an expendable feature. In this section, we review six categories of methods for reducing the size of a neural network.

Neural network compression aims to reduce the storage requirements of the network's weights. In [16, 42], low-rank approximation through matrix factorization, such as singular-value decomposition, is used to factorize the weight matrices. The factors' rank is reduced by keeping only the leading eigenvalues and their associated eigenvectors. In [24], quantization is used to reduce the storage taken by the model; both scalar quantization and vector quantization (VQ) have been considered. Using VQ, a weight matrix can be reconstructed from a list of indices and a dictionary of vectors. While network compression methods do not generally accelerate inference, some of them do, or can be adapted to do so. For example, when using VQ as in [24], one can avoid reconstructing the whole matrix with duplicate rows/columns, by using the codebook as the weight matrix and by altering the connectivity of the next layer. Thus, practical computation savings can be obtained. Unfortunately, most network compression methods do not decrease the memory and compute usage during inference.

Neural network pruning consists of identifying and removing neurons that are not necessary for achieving high performance. Some of the first approaches used the second-order derivative to determine the sensitivity of the network to the value of each weight [51, 32]. A more recent, very simple and effective approach selects which neurons to remove

by thresholding the magnitude of their weights; smaller magnitudes are associated with unimportant neurons [31]. The resulting network is then finetuned for better performance. Nonetheless, experimental results (c.f. Section 3.5) show that variational pruning methods (discussed below) outperform the previously mentioned works.

Sparsity Learning (SL) methods aim at pruning a network while training it. Some methods add to the training loss a regularization function such as L_1 [57], Group LASSO [97], or an approximation of the L_0 norm [58, 74]. Several variational methods have also been proposed [1, 48, 71, 67]. These methods formalize the problem of network pruning as a problem of learning the parameters of a dropout probability density function (PDF) via the reparametrization trick [48]. Pruning is enforced via a sparsity prior that derives from a variational evidence lower bound (ELBO). In general, SL methods do not apply an explicit constraint to limit the number of neurons used. To enforce a budget, one has to turn towards budgeted pruning.

Budgeted pruning is an approach that provides a direct control on the size of the pruned network via some “network size" hyper-parameter. MorphNet [26] alternates between training with a L_1 sparsifying regularizer and applying a width multiplier to the layer widths to enforce the budget. Contrary to our method, this work does not leverage dropout-based SL. Budgeted Super Networks [94] is a method that finds an architecture satisfying a resource budget by sparsifying a super network at the module level. This method is less convenient to use than ours, as it requires “neural fabric" training through reinforcement learning. Another budgeted pruning approach is “Learning-Compression" [7], which uses the method of auxiliary coordinates [6] instead of back-propagation. Contrary to this method, our approach adopts a usual gradient descent optimization scheme, and does not rely on the magnitude of the weights as a surrogate of their importance.

Architecture search (AS) is an approach that led to efficient neural networks in terms of performance and parameterization. Using reinforcement learning and vast amounts of processing power, NAS [103] have learned novel architectures; some that advanced the state-of-the-art, others that had relatively few parameters compared to similarly effective hand-crafted models. PNAS [54] and ENAS [77] have extended this work by cutting the necessary compute resources. These works have been aggregated by EPNAS [76]. AS is orthogonal to our line of work as the learned architectures could be pruned by our method. In addition, AS is more complicated to implement as it requires to learn a controller model by

reinforcement learning. In contrast, our method features tools widely used in CNN training.

3.3 Our Approach

3.3.1 Dropout Sparsity Learning

Before we introduce the specifics of our approach, let us first summarize the fundamental concepts of Dropout Sparsity Learning (DSL).

Let $\mathbf{h}_l$ be the output of the l-th hidden layer of a CNN computed by $f_l(\mathbf{h}_{l-1})$, a transformation of the previous layer, typically a convolution followed by a batch norm and a non-linearity. As mentioned before, one way of reducing the size of a network is by shutting down neurons with an element-wise product $\odot$ between the output of layer $\mathbf{h}_{l-1}$ and a binary tensor $\mathbf{z}_{l-1}$:

$$\mathbf{h}_l = f_i(\mathbf{h}_{l-1} \odot \mathbf{z}_{l-1}). \tag{3.1}$$

To enforce structured pruning and shutdown feature maps (not just individual neurons), one can redefine $\mathbf{z}_{l-1}$ as a vector of size d_{l-1} where d_{l-1} is the number of feature maps in $\mathbf{h}_{l-1}$. Then, $\mathbf{z}_{l-1}$ is applied over the spatial dimensions by performing an element-wise product with $\mathbf{h}_{l-1}$.

As one might notice, Eq. (3.1) is the same as that of dropout [69] for which $\mathbf{z}_{l-1}$ is a tensor of independent random variables *i.i.d.* of a Bernoulli distribution $q(z)$. To prune a network, DSL redefines $\mathbf{z}_{l-1}$ as random variables sampled from a distribution $q(z|\Phi)$ whose parameters Φ can be learned while training the model. In this way, the network can learn which feature maps to drop and which ones to keep.

Since the operation of sampling $\mathbf{z}_{l-1}$ from a distribution is not differentiable, it is common practice to redefine it with the reparametrization trick [48]:

$$\mathbf{h}_l = f_i(\mathbf{h}_{l-1} \odot g(\Phi_{l-1}, \epsilon)) \tag{3.2}$$

where g is a continuous function differentiable with respect to Φ and stochastic with respect to ϵ, a random variable typically sampled from $\mathcal{N}(0,1)$ or $\mathcal{U}(0,1)$.

In order to enforce network pruning, one usually incorporates a two-term loss :

$$L(W, \Phi) = L_D(W, \Phi) + \lambda L_S(\Phi) \tag{3.3}$$

where λ is the prior's weight, W are the parameters of the network, $L_D(W, \Phi)$ is a *data loss* that measures how well the model fits the training data (*e.g.* the cross-entropy loss) and L_S is a *sparsity loss* that measures how sparse the model is. While L_S varies from one method to another, the KL divergence between $q(z|\Phi)$ and some prior distribution is typically used by variational approaches [48, 67]. Note that during inference, one can make the network deterministic by replacing the random variable ϵ by its mean.

3.3.2 Soft and hard pruning

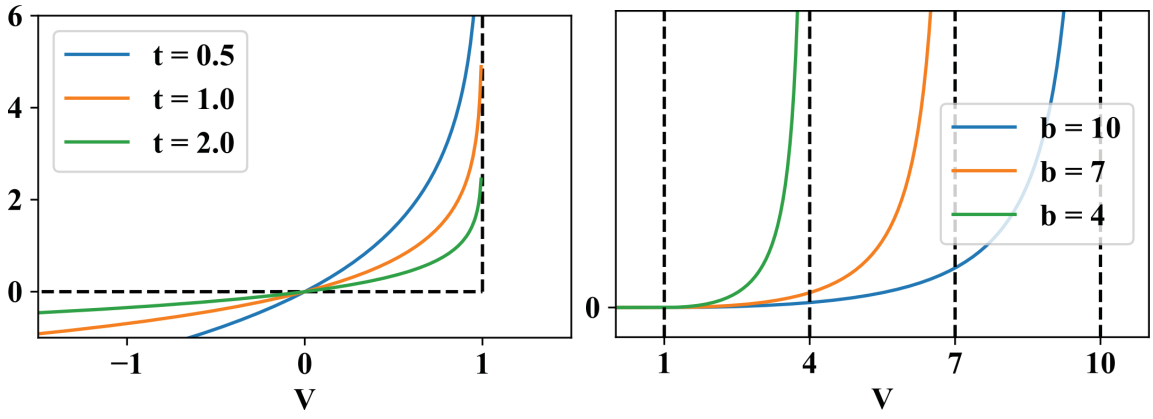
As mentioned before, $g(\Phi_{l-1}, \epsilon)$ is a continuous function differentiable with respect to Φ_{l-1}. Thus, instead of being binary, the pruning of Eq. (3.2) becomes continuous (soft pruning), so there is always a non-zero probability that a feature map will be activated during training. However, to achieve practical speedups, one eventually needs to "hard-prune" filters. To do so, once training is over, the values of Φ are thresholded to select which filters to remove completely. Then, the network may be fine-tuned for several epochs with the L_D loss only, to let the network adapt to hard-pruning. We call this the "fine-tuning phase", and the earlier epochs constitute the "training phase".

3.3.3 Budget-Aware Regularization

Our Budget-Aware Regularization (BAR) method integrates a budget constraint. In our implementation, a budget is the maximum number of neurons a "hard-pruned" network is allowed to have. To compute this metric, one may replace $z \sim q(z|\Phi)$ by its mean so feature maps with $\mathbb{E}[z|\Phi] = 0$ have no effect and can be removed, while the others are kept. The network size is thus the total activation volume of the structurally "hard-pruned" network :

$$V = \sum_l \sum_i \mathbb{1}(\mathbb{E}[z_{l,i}|\Phi] > 0) \times A_l \tag{3.4}$$

where A_l is the area of the output feature maps of layer l and $\mathbb{1}$ is the indicator function.



(a) Logarithmic barrier function

(b) Our barrier function

Figure 3.1 – **Comparing barrier functions.** *(a)* Common barrier function $-(1/t)\log(b - V)$ with $b = 1$. *(b)* Our barrier function $f(V, a, b)$ with $a = 1$.

A budget constraint imposes on V to be smaller than the allowed budget b. If embedded in a sparsity loss, that constraint makes the loss go to infinity when $V > b$, and zero otherwise. This is a typical inequality constrained minimization problem whose binary (and yet non-differentiable) behavior is not suited to gradient descent optimization. One typical solution to such problem is the *log-barrier* method [4]. The idea of this barrier method is to approximate the zero-to-infinity constraint by a differentiable logarithmic function : $-(1/t)\log(b-V)$ where $t > 0$ is a parameter that adjusts the accuracy of the approximation and whose value increases at each optimization iteration (c.f. Algo 11.1 in [4]).

Unfortunately, the log-barrier method requires to begin optimizing with a feasible solution (i.e. $V < b$), and this brings two major problems. First, we need to compute Φ such that $V < b$, which is no trivial task. Second, this induces a setting similar to training an ensemble of pruned networks, as the probability that a feature map is “turned on" is very low. This means that filters will receive little gradient and will train very slowly. To avoid this, we need to start training with a V larger than the budget.

We thus implemented a modified version of the barrier algorithm. First, as will be shown in the rest of this section, we propose a barrier function $f(V, a, b)$ as a replacement for the log barrier function (c.f. Fig. 3.1). Second, instead of having a fixed budget b and a parameter t that grows at each iteration as required by the barrier method, we eliminate the hardness parameter t and instead decrease the budget constraint at each iteration. This

budget updating schedule is discussed in Section 3.3.4.

Our barrier function $f(V, a, b)$ is designed such that:

— it has an infinite value when the volume used by a network exceeds the budget, *i.e.* $V > b$;

— it has a value of zero when the budget is *comfortably* respected, *i.e.* $V < a$;

— it has C^1 continuity.

Instead of having a jump from zero to infinity at the point where $V > b$, we define a range where a smooth transition occurs. To do so, we first perform a linear mapping of V :

$$c = \frac{V - a}{b - a}$$

such that $V = a \Rightarrow c = 0$ (the budget is *comfortably* respected), and $V = b \Rightarrow c = 1$ (our constraint $V < b$ is violated). Then, we use the following function:

$$g(c) = \frac{c^2}{1 - c}$$

which has three useful properties: (i) $g(0) = 0$ and $g(0)' = 0$, (ii) $\lim_{c \to 1^-} g(c) = \infty$ and (iii) it has a C^1 continuity. Those properties correspond to the ones mentioned before. To obtain the desired function, we substitute c in $g(c)$ and simplify:

$$f(V, a, b) = \begin{cases} 0 & V \leq a \\ \frac{(V-a)^2}{(b-V)(b-a)} & a < V < b \\ \infty & V \geq b. \end{cases} \tag{3.5}$$

As shown in Fig. 3.1, like for log barrier, $V = b$ is an asymptote, as we require $V < b$. However, $a < V < b$ corresponds to a respected budget and for $V \leq a$, the budget is respected with a comfortable margin, and this corresponds to a penalty of zero.

Our proposed prior loss is as follows:

$$L_{\text{BAR}}(\Phi, V, a, b) = L_S(\Phi) f(V, a, b) \tag{3.6}$$

where (a, b) are the lower and upper budget margins, V is the current "hard-pruned" volume as computed by Eq. (3.4), and $L_S(\Phi)$ is a differentiable approximation of V. Note

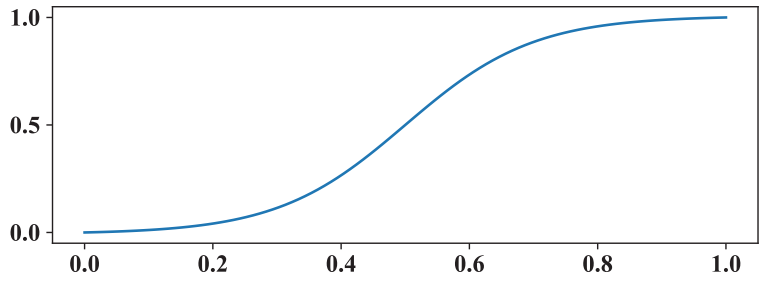


Figure 3.2 – **Sigmoidal transition function.**

that since V is not differentiable w.r.t to Φ, we cannot solely optimize $f(V, a, b)$.

The content of $L_S(\Phi)$ is bound to $q(z|\Phi)$. In our case, we use the Hard-Concrete distribution (which is a smoothed version of the Bernoulli distribution), as well as its corresponding prior loss, both introduced in [58]. This prior loss measures the expectation of the number of feature maps currently unpruned. To account for the spatial dimensions of the output tensors of convolutions, we use:

$$L_S(\Phi) = \sum_l L_S(\Phi_l) = \sum_l L_{HC}(\Phi_l) \times A_l$$

where L_{HC} is the hard-concrete prior loss [58] and A_l is the area of the output feature maps of layer l. Thus, $L_S(\Phi)$ measures the expectation of the activation volume of all convolution operations in the network.

Note that V could also be replaced by another metric, such as the total FLOPs used by the network. In this case, $L_S(\Phi_l)$ should also include the expectation of the number of feature maps of the preceding layer.

3.3.4 Setting the budget margins (a, b)

As mentioned earlier, initializing the network with a volume that respects the budget (as required by the barrier method) leads to severe optimization issues. Instead, we iteratively shift the pruning target b during training. Specifically, we shift it from $b = V_F$ at the beginning, to $b = B$ at the end (where V_F is the unpruned network's volume and B the

maximum allowed budget).

As shown in Fig. 3.1b, doing so induces a lateral shift to the “barrier". This is unlike the barrier method in which the hardness parameter t evolves in time (c.f. Fig. 3.1a). Mathematically, the budget b evolves as follows:

$$\begin{aligned} b &= (1 - T(i))\, V_F + T(i)\, B, \\ i &= \frac{\text{iteration index}}{\text{num. training iterations}} \end{aligned} \tag{3.7}$$

while $a = B - 10^{-4}\, V_F$ is fixed. Here $T(i)$ is a *transition function* which goes from zero at the first iteration all the way to one at the last iteration. While $T(i)$ could be a linear transition schedule, experimental results reveal that when b approaches B, some gradients suffers from extreme spikes due to the nature of $f(V, a, b)$. This leads to erratic behavior towards the end of the training phase that can hurt performance. One may also implement an exponential transition schedule. This could compensate for the shape of $f(V, a, b)$ by having b change quickly during the first epochs and slowly towards the end of training. While this gives good results for severe pruning (up to $16\times$), the increased stress at the beginning yields sub-optimal performance for low pruning factors.

For our method, we propose a sigmoidal schedule, where b changes slowly at the beginning and at the end of the training phase, but quickly in the middle. This puts most of the “pruning stress" in the middle of the training phase, which accounts for the difficulty of pruning (1) during the first epochs, where the filters' relevance is still unknown, and (2) during the last epochs, where more compromises might have to be made. The sigmoidal transition function is illustrated in Fig. 3.2 (c.f. Supp. materials for details).

3.3.5 Knowledge Distillation

Knowledge Distillation (KD) [37] is a method for facilitating the training of a small neural network (the *student*) by having it reproduce the output of a larger network (the *teacher*). The loss proposed by Hinton et al [37] is :

$$L_D(W) = (1 - \alpha) L_{CE}(P_s, Y_{gt}) + \alpha T^2 L_{CE}(P_s, P_t)$$

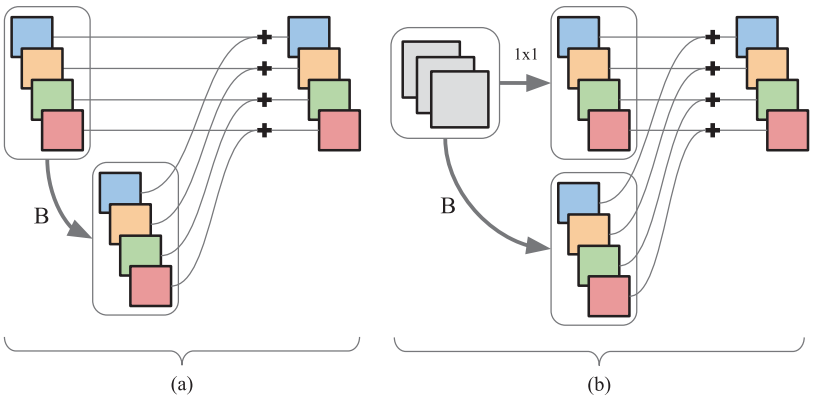


Figure 3.3 – **Typical ResBlock vs. pooling block.** *(a)* A typical ResBlock. The “B" arrow is the sequence of convolutions done inside the block. *(b)* A pooling block at the beginning of a ResNet Layer, that deals with the change in spatial dimensions and number of feature maps. Notice that it breaks the continuity of the residual signal. The arrow labeled “1×1" is a 1×1 convolution with stride 2; the first convolution of “B" also has stride 2. If all convolutions (arrows) are removed, no signal can pass.

where L_{CE} is a cross-entropy, Y_{gt} is the groundtruth, P_s and P_t are the output logits of the student and teacher networks, $\alpha \in [0, 1]$, and $T \geq 1$ is a temperature parameter used to smooth the softmax output : $p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$.

In our case, the unpruned network is the *teacher* and the pruned network is the *student*. As such, our final loss is:

$$(1 - \alpha)L_{CE}(P_s, Y_{gt}) + \alpha T^2 L_{CE}(P_s, P_t) + \lambda L_{\text{BAR}}(\Phi, V, a, b).$$

where λ, α and T are fixed parameters.

3.4 Pruning Residual Networks

While our method can prune any CNN, pruning a CNN without residual connections does not affect the connectivity patterns of the architecture, and simply selects the width at each layer [26]. In this paper, we are interested in allowing any feature map of a residual network to be pruned. This pruning regime can reduce the depth of the network, and generally results in architectures with atypical connectivity that require special care in their

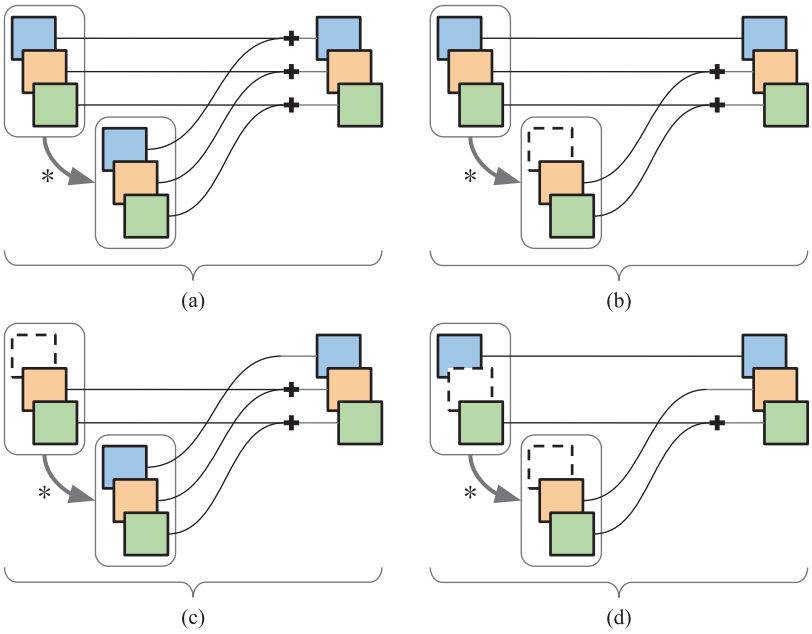


Figure 3.4 – **Connectivity allowed by our approach.** *(a)* A 3-feature ResBlock with typical connectivity. Arrows represent one or more convolutions. *(b)* With one feature map pruned, only two features are computed and added to the residual signal; one feature from the residual signal is left unchanged. *(c)* a new feature is created and concatenated to the residual signal. *(d)* a combination of (b) and (c) as a new feature is concatenated to the residual signal, one feature from the residual is left unchanged, and a third feature has typical connectivity (best viewed in color).

implementation to obtain maximum efficiency.

3.4.1 Automatic Depth Determination

We found, as in [26], that filter-wise pruning can successfully prune entire ResBlocks and change the network depth. This effect was named *Automatic Depth Determination* in [70]. Since a ResBlock computes a delta that is aggregated with the main (residual) signal by addition (c.f. Fig. 3.3a), such block can generally be removed without preventing the flow of signal through the network. This is because the main signal's identity connections cannot be pruned as they lack prunable filters.

However, some ResBlocks, which we call "pooling blocks", change the spatial dimen-

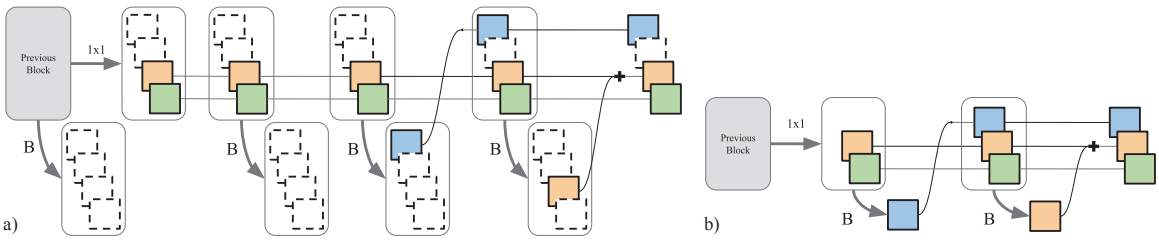


Figure 3.5 – (a) **A 4-feature chunk of a ResNet Layer pruned by our method**. Dotted feature maps are zeroed-out by their associated mask. An arrow labeled B represents a Block operation, which consist of a sequence of convolutions. Inner convolutions of the Block can be pruned, but only the output of the last convolution is shown (for clarity). (b) **The same pruned subgraph**, illustrated without the pruned feature maps. The resulting subgraph is shallower and narrower than its “full" counterpart (best viewed in color).

sions and feature dimensionality of the signal. This type of block breaks the continuity of the residual signal (c.f. Fig. 3.3b). As such, the convolutions inside this block cannot be completely pruned, as this would prevent any signal from flowing through it (a situation we call “fatal pruning"). As a solution, we clamp the highest value of Φ so at least one feature map is kept in the 1×1 conv operation.

3.4.2 Atypical connectivity of pruned ResNets

Our method allows any feature map in the output of a convolution to be pruned (except for the 1×1 conv of the pooling block). This produces three types of atypical residual connectivity that requires special care (see Fig. 3.4). For example, there could be a feature from the residual signal that would pass through without another signal being added to it (Fig. 3.4b). New feature maps can also be created and concatenated (Fig. 3.4c). Furthermore, new feature maps could be created while others could pass through (Fig. 3.4d).

To leverage the speedup incurred by a pruned feature map, the three cases in Fig. 3.4 must be taken into account through a mixed-connectivity block which allows these unorthodox configurations. Without this special implementation, some zeroed-out feature maps would still be computed because the summations of residual and refinement signals must have the same number of feature maps. In fact, a naive implementation does not allow to refine only a subset of the features of the main signal (as in Fig. 3.4b), nor does it allow to

have a varying number of features in the main signal (as in Fig. 3.4c).

Fig. 3.5 shows the benefit of a mixed-connectivity block. In (a) is a ResNet Layer pruned by our method. Using a regular ResBlock implementation, all feature maps in pairs of tensors that are summed together need to have matching width. This means that, in Fig. 3.5, all feature maps of the first, third and fourth rows are computed, even if they are dotted. Only the second row can be fully removed; however, there are no restrictions on pruning in inner convolutions of ResBlocks. On the other hand, by using mixed-connectivity, only unpruned feature maps are computed, yielding architectures such as in Fig. 3.5b, that saves substantial compute (c.f. Section 3.5).

Technical details on our mixed-connectivity block are provided in the Supplementary materials.

3.5 Experiments

3.5.1 Experimental Setup

We tested our pruning framework on two residual architectures and report results on three datasets. We pruned Wide-ResNet [100] on CIFAR-10 and CIFAR-100 (with a width multiplier of 12 as per [100]), and ResNet50 [35] on Mio-TCD [61], a larger and more complex dataset devoted to traffic analysis. The ResNet50 is fed with images of size 128×128 as opposed to 32×32 for Wide-ResNet. Since this ResNet50 has a larger input and is deeper than its CIFAR counterpart, we do not opt for the “wide" version and thus save significant training time. Both network architectures have approximately the same volume.

For all experiments, we use the Adam optimizer with an initial learning rate of 10^{-3} and a weight decay of 5×10^{-4}. For CIFAR-10 and CIFAR-100, we use a batch size of 64. For our objective function, we use $\alpha = 0.9$, $T = 4$, and $\lambda = 10^{-5}$. We use PyTorch and its standard image preprocessing. For experiments on Mio-TCD, we start training/pruning with the weights of the unpruned network whereas we initialize with random values for CIFAR-10 and CIFAR-100. Please refer to the Supplementary materials for the number of epochs used in each training phase.

We compare our approach to the following methods:

— **Random**. This approach randomly selects feature maps to be removed.

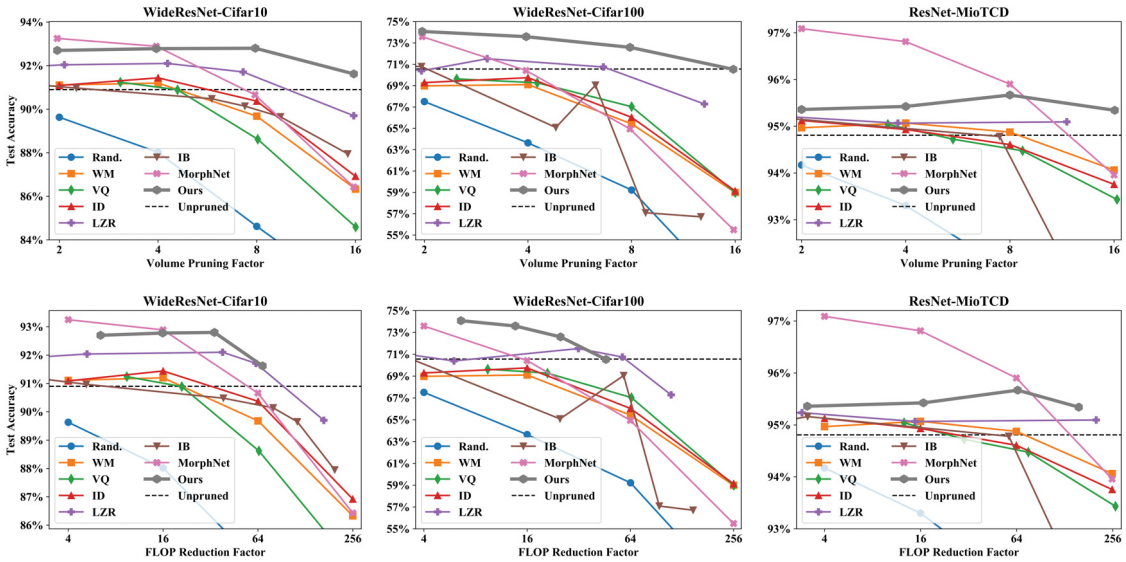


Figure 3.6 – **Pruning results.** Plots showing test accuracy w.r.t. volume and FLOP reduction factor (best viewed in color).

— **Weight Magnitude (WM)** [31]. This method uses the absolute sum of the weights in a filter as a surrogate of its importance. Lower magnitude filters are removed.

— **Vector Quantization (VQ)** [24] This approach vectorizes the filters and quantizes them into N clusters, where N is the target width for the layer. The clusters' center are used as the new filters.

— **Interpolative Decomposition (ID)**. This method is based on low-rank approximation for network compression [16, 42]. This algorithm factorizes each filters W into UV, where U has a specific number of rows corresponding to the budget. U replaces W, and V is multiplied at the next layer (i.e. $W_{l+1} \leftarrow V_l W_{l+1}$) to approximate the original sequence of transformations.

— L_0 **regularization (LZR)** [58]. This DSL method is the closest to our method. However, it incorporates no budget, penalizes layer width instead of activation tensor volume, and does not use Knowledge Distillation.

— **Information Bottleneck (IB)** [1]. This DSL method uses a factorized Gaussian distribution (with parameters μ, σ) to mask the feature maps as well as the following prior loss : $L_S = \log(1 - \mu^2/\sigma^2)$.

Table 3.1 – **Test Accuracy for different configurations of our method** (using WideResNet-CIFAR-10). The test accuracy of the unpruned network is 90.90%.

Configuration	**Pruning factor**	
	2x	16x
Our method	92.70%	91.62%
w/o Knowledge Distillation	-1.37%	-0.40%
w/o Sigmoid pruning schedule	-0.87%	-0.92%

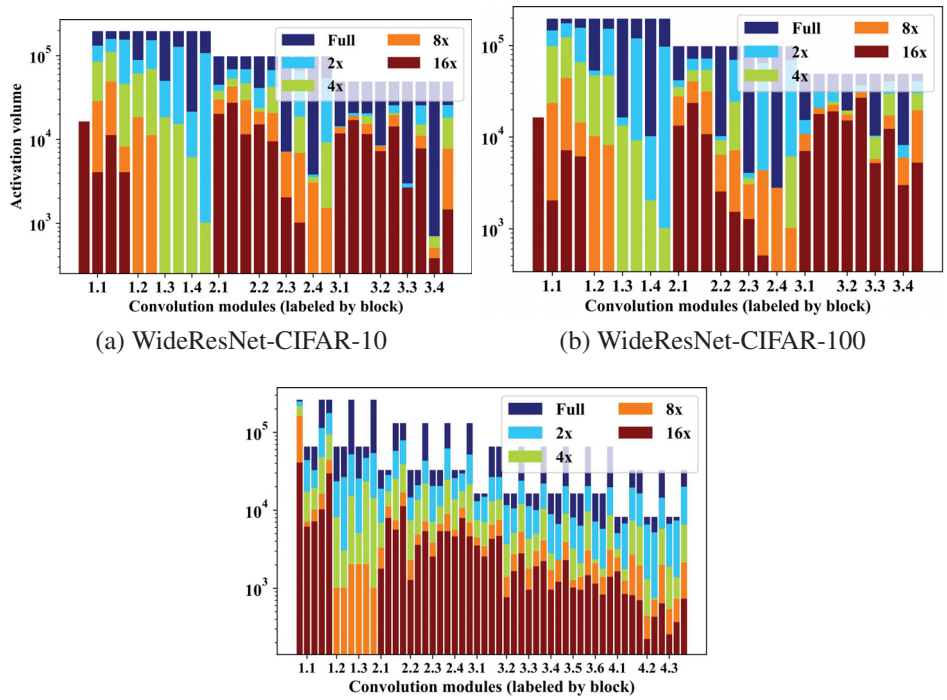
— **MorphNet** [26]. This approach uses the γ scaling parameter of Batch Norm modules as a learnable mask over features. The said γ parameters are driven to zero by a L_1 objective that considers the resources used by a filter (e.g. FLOPs). This method computes a new width for each layer by counting the non-zero γ parameters. We set the sparsity trade-off parameter λ after an hyperparameter search, with $16\times$ as the target pruning factor for CIFAR-10.

For every method, we set a budget of tensor activation volume corresponding to $1/2$, $1/4$, $1/8$, $1/16$ of the unpruned volume V_F. Since *LZR* and *IB* do not allow to set a budget, we went through trial-and-error to find the hyperparameter value that yield the desired resource usage. For *Random, WM, VQ*, and *ID* we scale the width of all layers uniformly to satisfy the budget and implement a pruning scheme which revealed to be the most effective (c.f. Supplementary materials). We also apply our proposed mixed-connectivity block to the output of every method for a fair comparison.

3.5.2 Results

Results for every method executed on all three datasets are shown in Fig. 3.6. The first row shows test accuracies w.r.t. the network volume reduction factor for CIFAR-10, CIFAR-100 and Mio-TCD. As one can see, our method is above the others (or competitive) for CIFAR-10 and CIFAR-100. It is also above every other method on Mio-TCD except for MorphNet which is better for pruning factors of 2 and 4. However, MorphNet gets a severe drop of accuracy at 16x, a phenomena we observed as well on CIFAR-10 and CIFAR-100. Our method is also always better than IB and LZR, the other two DSL methods. Overall, our method is also resilient to severe (16x) pruning ratios.

Furthermore, for every dataset, networks pruned with our method (as well as some



(a) WideResNet-CIFAR-10

(b) WideResNet-CIFAR-100

(c) ResNet-Mio-TCD

Figure 3.7 – **Result of pruning with our method.** Total number of active neurons in the full networks and with four different pruning rates. Positions without an orange (8x) or red (16x) bar are those for which a Resblock has been eliminated.

Table 3.2 – **Reduction of the effective pruned volume when removing the mixed-connectivity block.**

Dataset	2x	4x	8x	16x
CIFAR-10	12%	43%	53%	58%
CIFAR-100	14%	49%	55%	57%
MIO-TCD	32%	37%	40%	52%

others) get better results than the initial unpruned network. This illustrates the fact that Wide-ResNet and ResNet-50 are too-large networks for certain tasks and that decreasing their number of feature maps reduces overfitting and thus improves test accuracy.

We then took every pruned network and computed their FLOP reduction factor (we considered operations from convolutions only). This is illustrated in the second row of Fig. 3.6. There again, our method outperforms (or is competitive with) the others for CIFAR-10 and CIFAR-100. Our method reduces FLOPs by up to a factor of ~ 64x on CIFAR-10, ~ 60x on CIFAR-100 and ~ 200x on Mio-TCD without decreasing test accuracy. We get similar results as LZR for pruning ratios around 60x on CIFAR-10 and CIFAR-100 and 200x on Mio-TCD. MorphNet gets better accuracy for pruning ratios of 4x and 16x on Mio-TCD, but then drops significantly around 256x.

We also compare the performance of our method when removing some of its components. In Table 3.1, we perform these experiments on WideResNet-CIFAR-10 with two pruning factors. We first replaced the Knowledge Distillation data loss (c.f. Section 3.3.5) by a cross-entropy loss, and also changed the Sigmoid pruning schedule (c.f. Section 3.3.4) by a linear one. As can be seen, removing either of those reduces accuracy, thus showing their efficiency. We also studied the impact of not using the mixed-connectivity block introduced in Section 3.4.2. As shown in Table 3.2, when replacing our mixed-connectivity blocks by regular ResBlocks, we get a drop of the effective pruned volume of more than 50% for 16x (even up to 58% for CIFAR-10).

We illustrate in Fig. 3.7 the result of our pruning method all datasets. The figure shows the number of neurons per residual block for the full network, and for the networks pruned with varying pruning factors. These plots show that our method has the capability of eliminating entire residual blocks (especially around 1.3 and 1.4 for CIFAR-10 and 100). Also, the pruning configurations follow no obvious trend thus showing the inherent plasticity of

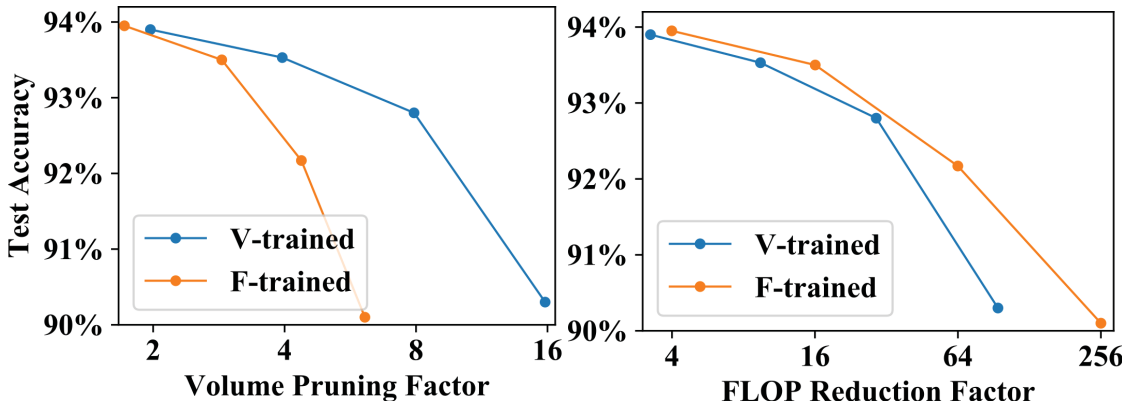


Figure 3.8 – **Comparison of objective metrics.** Test accuracy versus the volume pruning factor and the FLOP reduction factor for our method with a Volume metric (V-trained) and a FLOP metric (F-trained).

a DSL method such as ours.

As mentioned in Section 3.3.3, instead of the volume metric (Eq. (3.4)) the budget could be set w.r.t a FLOP metric by accounting for the expectation of the number of feature maps in the preceding layer. We compare in Fig. 3.8 the results given by these two budget metrics for WideResnet-CIFAR-10. As one might expect, pruning a network with a volume metric (*V-Trained*) yields significantly better performances w.r.t. the volume pruning factor whereas pruning a network with a FLOP metric (*F-Trained*) yields better performances w.r.t. to the FLOP reduction factor, although by a slight margin. In light of these results, we conclude that the volume metric (Eq. (3.4)) is overall a better choice.

3.6 Conclusion

We presented a structured budgeted pruning method based on a dropout sparsity learning framework. We proposed a knowledge distillation loss function combined with a budget-constrained sparsity loss whose formulation is that of a barrier function. Since the log-barrier solution is ill-suited for pruning a CNN, we proposed a novel barrier function as well as a novel optimization schedule. We provided concrete insights on how to prune residual networks and used a novel mixed-connectivity block. Results obtained on two ResNets architecture and three datasets reveal that our method overall outperforms 7 other pruning methods.

Conclusion

Cet ouvrage a présenté des moyens efficaces pour faciliter et améliorer l'analyse du trafic routier. L'analyse d'images captées avec des caméras routières est avantageuse, et les CNN permettent d'en tirer un maximum. Les CNN sont des réseaux neuronaux capables d'apprendre des motifs sous forme de structures spatiales ; le début de cet ouvrage en a présenté les fondements et les principes d'entraînement. De plus, la base de données MIO-TCD, dont le candidat a participé à la publication, permet d'entraîner ces CNN sur les tâches de classification et de localisation de véhicules routiers. Bien que les CNN soient hautement efficaces pour les tâches à accomplir, ils ne sont généralement pas assez efficients pour être exécutés à même une caméra (ce qui permettrait de réduire la bande passante en transmettant seulement les données extraites).

Pour réduire le fardeau computationnel des CNN, l'élagage a été présenté. L'élagage permet d'optimiser la structure d'un réseau neuronal, pour en éliminer les neurones qui n'affectent pas sa performance. Différentes approches d'élagages ont été abordées, dont l'apprentissage de parcimonie, et l'élagage avec budget. Enfin, le candidat a présenté une nouvelle méthode combinant l'apprentissage de parcimonie et l'élagage avec budget. Cette méthode surpasse l'état de l'art pour les degrés de parcimonie les plus sévères. Une direction possible d'amélioration serait d'enrichir l'espace des structures possibles en incluant une variété de sous-structures ne pouvant pas être toutes présentes simultanément dans un seul réseau à élaguer ; cette approche nécessiterait d'échantillonner les neurones à utiliser à chaque itération.

Il reste de nombreuses questions méritant d'être étudiées dans des projets de recherche futurs. L'optimisation de la structure d'un réseau neuronal est un méta-problème d'apprentissage, qui requiert encore un bon nombre d'hyperparamètres et d'essai-erreur. Les phénomènes qui permettent ou non l'émergence d'un neurone utile ne sont pas bien com-

pris. La taille d'un ensemble de données, la difficulté de la tâche, le temps d'entraînement et la structure du réseau sont parmi un nombre d'éléments interdépendants qui influencent l'efficience d'un réseau neuronal. Des travaux permettant de d'extraire du sens de ce chaos apparent seraient d'une grande utilité pour le domaine.

L'élagage applique une pression sur les ressources utilisées par les réseaux de neurones artificiels, de manière similaire aux contraintes d'énergie qui sont imposées aux réseaux neuronaux biologiques. Puisque ces réseaux biologiques semblent éviter remarquablement le gaspillage, il paraît logique que des caractéristiques tirées d'un ensemble limité de neurones artificiels soient plus faciles à interpréter par le cerveau humain que leurs contreparties tirées d'un modèle surparamétrisé. La confirmation de cette hypothèse s'intègrerait à la lignée des travaux sur l'interprétabilité des caractéristiques dans les réseaux neuronaux artificiels [73, 72, 68].

Appendix A

MIO-TCD: A New Benchmark Dataset for Vehicle Classification and Localization

Cet article présente une nouvelle base de données : MIO-TCD. Cette base de données a été mise sur pied pour stimuler la recherche pour les tâches de classification et de localisation de véhicules dans les images de trafic routier. Elle contient un 786 702 images, dont 648 959 pour la tâche de classification et 137 743 pour la tâche de localisation. Il s'agit lors de sa publication du plus grand ensemble de données du genre. La tâche de classification consiste à identifier la catégorie d'un véhicule parmi 11 catégories : *Articulated truck, Bicycle, Bus, Car, Motorcycle, Non-motorized vehicle, Pedestrian, Pickup truck, Single unit truck, Work van, Background*. Il peut y avoir 0 ou 1 véhicule dans l'image. Quant à elle, la tâche de localisation consiste à identifier 0, 1 ou plusieurs véhicules, en indiquant pour chaque objet la catégorie et les paramètres du rectangle englobant. En plus de présenter la base de données et les tâches, l'article présente des résultats de classification et de localisation obtenus avec les méthodes de l'état de l'art. Ces résultats tendent à démontrer la viabilité des CNN profonds pour les tâches de classification et de localisation de véhicules. Une analyse des cas d'échec des méthodes actuelles est aussi présentée, avec des suggestions pour y remédier.

Le candidat a participé à l'élaboration de cet article en implémentant et/ou entraînant six modèles de CNN pour la tâche de classification MIO-TCD. Le candidat a aussi participé

à la révision du texte et à la création des figures.

Contributions

Les contributions au domaine présentées dans l'article sont listées ci-dessous.

— Mise sur pied d'une base de données riche, servant de banc d'essai pour comparer les méthodes de classification et de localisation de véhicules ;

— Mise sur pied de tableaux de classement et de comparaison des meilleures méthodes, disponible sur le web : http://tcd.miovision.com ;

— Identifier les cas d'échec propices à être étudiés dans des travaux de recherche futurs.

L'article a été publié dans le journal IEEE Transactions on Image Processing (volume 27, numéro 10, octobre 2018 – disponible en ligne dès le 18 juin 2018).

MIO-TCD: A New Benchmark Dataset for Vehicle Classification and Localization

Zhiming Luo[1,2], Frédéric B.-Charron[2], Carl Lemaire[2], Janusz Konrad[3], Shaozi Li[1], Akshaya Mishra[4], Andrew Achkar[5], Justin Eichel[5], and Pierre-Marc Jodoin[2]

[1]*Xiamen University, Xiamen, China*
[2]*University of Sherbrooke, Sherbrooke, Canada*
[3]*Boston University, Boston, USA*
[4]*University of Waterloo, Waterloo, Canada*
[5]*Miovision Technologies Inc., Kitchener, Canada*

Abstract

The ability to train on a large dataset of labeled samples is critical to the success of deep learning in many domains. In this paper, we focus on motor vehicle classification and localization from a single video frame and introduce the "MIOvision Traffic Camera Dataset" (MIO-TCD) in this context. MIO-TCD is the largest dataset for motorized traffic analysis to date. It includes 11 traffic object classes such as cars, trucks, buses, motorcycles, bicycles, pedestrians. It contains 786,702 annotated images acquired at different times of the day and different periods of the year by hundreds of traffic surveillance cameras deployed across Canada and the United States. The dataset consists of two parts: a "localization dataset", containing 137,743 full video frames with bounding boxes around traffic objects, and a "classification dataset", containing 648,959 crops of traffic objects from the 11 classes. We also report results from the 2017 CVPR MIO-TCD Challenge, that leveraged this dataset, and compare them with results for state-of-the-art deep learning architectures. These results demonstrate the viability of deep learning methods for vehicle localization and classification from a single video frame in real-life traffic scenarios. The top-performing methods achieve both accuracy and Kappa score above 96% on the classification dataset and mean-average precision of 77% on the localization dataset. We also identify scenarios in which state-of-the-art methods still fail and we suggest avenues to address these challenges. Both the dataset and detailed results are publicly available on-line [63].

A.1 Introduction

The localization and classification of vehicles in the field of view of a traffic camera is the very first step in most traffic surveillance systems (e.g., car counting, activity recognition, anomaly detection, tracking, post-event forensics). Although subsequent processing may be different in each case, one often has to start by localizing foreground objects and then identifying what these objects are (trucks, cars, bicycles, pedestrians, etc.).

To the best of our knowledge, except for pedestrian applications, most traffic monitoring systems rely on motion features such as optical flow, motion detection, and vehicle tracking [96, 101, 55]. Motion features are then used to count the number of vehicles on the road, estimate traffic speed, and recover global motion trajectories. But these traffic monitoring systems all share a fundamental limitation: in order to function properly, they need high frame-rate videos so motion features can be reliably extracted [45]. Also, reliable object tracking is still a challenge for cluttered scenes, especially when vehicles are partly occluded [60].

Unfortunately, low frame-rate videos are common as many large-scale camera networks cannot stream and store high frame-rate videos gathered by thousands of cameras. Instead, cameras are often configured to send one frame every second or so to the server. This is especially true for cameras transmitting over a cellular network whose bandwidth may vary over time making the throughput unpredictable. In such cases, one can only analyze ultra low frame-rate videos out of which no motion features can be accurately extracted. Also, those cameras often have low resolution which makes localization and classification difficult.

To date, many object localization algorithms have been developed, and even more articles have been written on the topic. With the rise of deep learning methods (mostly convolutional neural nets), an exponential number of publications have been devoted to deep architectures implementing object recognition and localization methods [23, 22, 82, 36]. As a result, error rates on very challenging datasets, such as Pascal VOC [19], ImageNet [85] and Microsoft COCO [53], have decreased at a steady pace. Among other things, what makes deep learning so successful is the availability of large and well-annotated datasets. Unfortunately, despite the large number of publications devoted to traffic analytics, no such traffic dataset has been released to date. The lack of such a dataset has a number of implica-

tions, the most important one being that deep architectures trained on non-traffic oriented datasets such as ImageNet and COCO do not generalize well to traffic images.

Recognizing the importance of labeled datasets for the development of traffic analysis methods, we introduce the *MIOvision Traffic Camera Dataset* (MIO-TCD). It contains a total of 786,702 images: 137,743 high-resolution video frames with multiple vehicles in each frame and 648,959 vehicles cropped out of full frames. These images have been captured by hundreds of cameras deployed in urban and rural areas taking pictures at different periods of the year, at different times of the day, with different camera orientations and with various traffic densities. Many people have participated in the process of hand-annotating each image to provide a bounding box around each distinguishable object.

Leveraging this dataset, we demonstrate how well-trained, state-of-the-art deep learning methods can be used to localize and identify moving objects with high precision without having to rely on motion features. With classification accuracies of 96% and localization mean-average precision of 77%, we show that one can localize and recognize vehicles regardless of their orientation, scale, color and environmental conditions.

We believe that this work will have a substantial impact as there exists an urgent need for traffic monitoring systems working on still 2D images. We hope the MIO-TCD datset will have similar impact on the video surveillance community as the Caltech [17] and INRIA [14] datasets had on the pedestrian detection community, or ImageNet and COCO datasets – on the deep learning community.

The MIO-TCD dataset was the foundation of the Traffic Surveillance Workshop and Challenge held in conjunction with CVPR 2017. There were three primary goals to this challenge:

1. Provide the scientific community with a rich dataset to compare and test new methods (the dataset will be regularly revised and expanded in the future and will maintain a ranking of methods).
2. Identify and rank the best traffic-oriented object localization and classification algorithms to date in various real-life conditions.
3. Identify remaining critical challenges in order to provide focus for future research.

In what follows, we elaborate on these goals, describe the dataset and the challenge results, and discuss unsolved challenges while offering avenues for future work.

A.2 Previous datasets

Today, we are witnesses to an exponential growth in the number of surveillance cameras are deployed along the roads of almost every country in the world. However, the images acquired by those cameras are the sole property of traffic management departments and are rarely released to the public. Consequently, few datasets have been made public for traffic analysis research. The most widely used datasets can be roughly divided into 3 categories, namely: (1) datasets of images taken by on-board cameras and mainly aimed at autonomous driving, (2) datasets of vehicle images taken by non-surveillance cameras and mainly oriented towards automatic recognition of high-resolution images from the Internet and (3) datasets of vehicle images taken by surveillance cameras. Here, we present a brief survey of existing vehicle detection and localization datasets.

On-board camera datasets

KITTI benchmark Dataset [21] [1]: This is a large dataset collected by an autonomous driving platform which addresses several real-world challenges, including: stereo vision calculation, optical flow estimation, visual odometry/SLAM, 3D object detection and 3D object tracking. This dataset consists of video frames captured by cars traveling both in rural areas and on highways.

Cityscapes Dataset [11] [2]: This dataset focuses on the semantic segmentation of urban street scenes. It comes with 5,000 fully-annotated images and 20,000 weekly-annotated images. The annotated objects span 30 classes including eight different types of vehicles. Unfortunately, this dataset is limited to images acquired in urban areas (50 cities) and during summer daytime.

Tsinghua-Tencent Traffic-Sign Dataset [102] [3]: This is a large traffic-sign dataset containing 100,000 images with 30,000 traffic-sign instances. Images in this dataset cover various illumination and weather conditions but do not contain any labeled vehicles.

1. http://www.cvlibs.net/datasets/kitti/index.php
2. https://www.cityscapes-dataset.com/
3. http://cg.cs.tsinghua.edu.cn/traffic-sign/

Vehicles captured by non-surveillance cameras

Stanford Car Dataset [49] [4]: This dataset contains 16,185 high-resolution images of 196 classes of cars. The vehicle classes include the brand, the model, and the year (e.g. 2012 Tesla Model S or 2012 BMW M3 coupe). This dataset is divided into 8,144 training images and 8,041 testing images. In addition to the large variety of vehicles, all pictures have excellent resolution and have been captured in good lighting conditions. However, none of the pictures were taken in a top-down orientation which is usually the case with surveillance cameras.

Comprehensive Cars Dataset (web) [98] [5]: This is one of the largest car dataset currently available. It comes with a total of 136,727 images showing entire cars and 27,618 images showing car parts. The dataset comprises 1,716 vehicle models as well as five different attributes (maximum speed, displacement, number of doors, number of seats, and type of car). Unfortunately, this dataset has the same limitations as the Stanford Car Dataset as its images were taken in a context far different than that of a surveillance camera (arbitrary orientation and filming 24/7).

Traffic datasets from surveillance cameras

Comprehensive Cars Dataset (surveillance) [98]: This dataset contains 44,481 images of cars captured by frontal-view surveillance cameras. The ground truth provides the model and the color of each vehicle. The main limitation of this dataset comes from the frontal view of the pictures which makes it hard to generalize to an arbitrary camera orientation. Furthermore, this dataset focuses on cars, mini vans and pickup trucks, and does not contain any large articulated trucks, buses, motorcycles and pedestrians.

BIT-Vehicle Dataset [18] [6]: This dataset consists of 9,850 vehicle images. This is one of the most realistic datasets with images coming from real surveillance cameras. Vehicles are divided into six categories, namely bus, micro-bus, minivan, sedan, SUV and truck. Unfortunately, those images were all taken in a top-frontal view during daytime and clear weather, thus limiting the diversity needed for general traffic analysis applications.

4. http://ai.stanford.edu/ jkrause/cars/car_dataset.html
5. http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html
6. http://iitlab.bit.edu.cn/mcislab/vehicledb/

Traffic and Congestions (TRANCOS) Dataset [28][7]: This dataset is used to count the number of vehicles on highly-congested highways. It consists of 1,244 images with 46,796 annotated vehicles, most of which are partially occluded. Images were captured by publicly available video surveillance cameras of the Dirección General de Tráfico of Spain. Unfortunately, no vehicle type is provided.

GRAM Road-Traffic Monitoring (GRAM-RTM) Dataset [27][8]: This is a benchmark dataset for multi-vehicle tracking. It consists of video sequences recorded by surveillance cameras under different conditions and with different platforms. Every vehicle has been manually annotated into different categories (car, truck, van, and big truck). Each video contains around 240 different objects.

Figure A.1 – Sample images from the 11 categories of the classification dataset.

Clearly, several publicly-available vehicle datasets contain images that do not come from surveillance cameras. The *KITTI benchmark dataset*, the *Cityscapes Dataset* and the *Tsinghua-Tencent Traffic-Sign Dataset* contain images captured by on-board cameras that can hardly be used to train traffic surveillance methods. The *Stanford Car Dataset* and the *Comprehensive Car Dataset (web)* contain high-resolution pictures of vehicles mainly taken in frontal and side view and rarely in top-down view as is usually the case with traffic surveillance applications. Images from these datasets are usually applied to fine-grained vehicle analysis (counting the number of doors, identifying the vehicle brand and year, etc.). As for the *Comprehensive Cars Dataset (surveillance)*, although it is one of the largest publicly-available surveillance dataset, its images come from a well-aligned frontal-view camera and show only one vehicle per image. It also shows images in daylight and

7. http://agamenon.tsc.uah.es/Personales/rlopez/data/trancos/
8. http://agamenon.tsc.uah.es/Personales/rlopez/data/rtm/

good weather. Also, none of the datasets contains more than a couple of thousand images.

As for the *BIT-Vehicle Dataset*, it has up to two vehicles per frame but contains less than 10,000 images. The *TRANCOS Dataset* contains traffic jam images in which vehicles are too small to be categorized while the *GRAM-RTM Dataset* has only three video sequences.

A.3 MIO-TCD : Our Proposed Dataset

A.3.1 Dataset Overview

The MIO-TCD dataset contains a total of 786,702 images, 137,743 being high-resolution video frames showing multiple vehicles and 648,959 lower-resolution images of cropped vehicles. These images were acquired at different times of the day and different periods of the year by hundreds of traffic cameras deployed across Canada and the United States. Those images have been selected to cover a wide range of challenges and are typical of visual data captured in urban and rural traffic scenarios. The dataset includes images: (1) taken at different times of the day, (2) with various levels of traffic density and vehicle occlusion, (3) showing small moving objects due to low resolution and/or perspective, (4) showing vehicles with different orientations, (5) taken under challenging weather conditions, and (6) exhibiting strong compression artifacts. This dataset has been carefully annotated by a team of nearly 200 people to enable a quantitative comparison and ranking of various algorithms.

The MIO-TCD dataset aims to provide a rigorous facility for measuring how far state-of-the-art deep learning methods can go at classifying and localizing vehicles recorded by traffic cameras. The dataset consists of two components: the *classification dataset* and the *localization dataset*. The classification dataset is used to train and test classification algorithms whose goal is to predict the kind of vehicle located in a low-resolution image patch. The localization dataset may be used to train and test algorithms whose goal is to localize and recognize vehicles located in the image.

MIO-TCD - Classification Dataset

The classification dataset contains 648,959 low-resolution images divided into 11 categories: *Articulated Truck*, *Bicycle*, *Bus*, *Car*, *Motorcycle*, *Non-Motorized Vehicle*, *Pedes-*

trian, *Pickup Truck*, *Single-Unit Truck*, *Work Van* and *Background*. As shown in Fig. A.1, each image contains one dominant object located in the middle of the image. The objects pictured in that dataset come in various sizes and were recorded in different parts of the year, different times of the day and from different viewing angles. The dataset was split into 80% training (519,164 images) and 20% testing (129,795 images). Note that the *Car* category contains vehicles of type sedan, SUV and family van.

Figure A.2 – Sample images from the MIO-TCD localization dataset.

The number of images in each category is listed in Table A.1. Since these images come from real footage, the number of samples per category is highly unbalanced as certain types of vehicles are more frequent than others. As such, almost half the images in the dataset fall into the *car* category, while the *Bicycle*, *Motorcycle* and *Non-Motorized Vehicle* categories contain roughly 2,000 training images and 500 testing images. We also randomly sampled 200,000 images to construct a background category.

MIO-TCD - Localization Dataset

The localization dataset contains 137,743 images of which 110,000 are intended for training and 27,743 for testing. These images have various resolutions, ranging from 720×480 to 342×228. A total of 416,277 moving objects have been manually annotated with a bounding box and a category label. Except for *Background*, the same category labels as those in Table A.1 have been used. However, due to the fact that localizing and recognizing vehicles in a full video frame is more difficult than classifying images of already localized vehicles, we added a new *Motorized Vehicle* category which is unique to the localization

Table A.1 – Size of each category in the classification dataset

Category	**Training**	**Testing**
Articulated Truck	10,346	2,587
Bicycle	2,284	571
Bus	10,316	2,579
Car	260,518	65,131
Motorcycle	1,982	495
Non-Motorized Vehicle	1,751	438
Pedestrian	6,262	1,565
Pickup Truck	50,906	12,727
Single-Unit Truck	5,120	1,280
Work Van	9,679	2,422
Background	160,000	40,000
Total	**519,164**	**129,795**

dataset. This category contains all vehicles that are too small (or occluded) to be labeled into a specific category. Because of this category overlap, the classification dataset can be leveraged to improve the accuracy of localization methods.

Similarly to the classification dataset, images of the localization dataset came from hundreds of real traffic surveillance cameras. The category labels are thus unbalanced in similar proportions as those in Table A.1 (see Fig. A.2 for some examples).

A.3.2 Evaluation Metrics

Classification

Three metrics have been implemented to gauge performance of classification methods. The first metric is the overall accuracy (Acc) which is the proportion of correctly classified images in the whole dataset:

$$Acc = \frac{TP}{\text{total number of images}} \tag{A.1}$$

where TP is the total number of correctly-classified images regardless of their category. TP can also be seen as the trace of a confusion matrix such as the one in Fig. A.3.

Since the classification dataset is highly unbalanced, large categories such as *Car* and *Background* have an overwhelming influence on the calculation of the overall accuracy. We thus implemented three metrics which account for this imbalance, namely the mean recall (mRe), the mean precision (mPr) and the Cohen Kappa Score ($Kappa$) [43].

The mean recall and mean precision are obtained by averaging the recall and the precision of each category thus giving an equal weight to each category. This is done as follows:

$$mRe = \frac{\sum_{i=1}^{11} Re_i}{11} \qquad mPr = \frac{\sum_{i=1}^{11} Pr_i}{11}$$

where $Re_i = TP_i/(TP_i + FN_i)$ and $Pr_i = TP_i/(TP_i + FP_i)$ are the recall and precision for category i, and TP_i, FN_i, FP_i are the numbers of true positives, false negatives and false positives for the i-th category, respectively.

$Kappa$ is a measure that expresses the agreement between two annotators. In our case, the first annotator is a method under evaluation and the second annotator is the ground truth. The Cohen Kappa Score is defined as [43]:

$$Kappa = \frac{Acc - P_e}{1 - P_e} \tag{A.2}$$

where Acc is the accuracy (A.1) and P_e is the probability of agreement when both annotators assign random labels. $Kappa$ values are in the $[-1, 1]$ range where $Kappa = 1$ means that both annotators are in complete agreement, while $Kappa \leq 0$ means no agreement at all.

Localization

Following the Pascal VOC 2012 object detection evaluation protocol, we report localization results *via* precision/recall curves and use the average precision (AP) as the principal quantitative measure for each vehicle category. A detection is considered a true positive when the overlap ratio between the predicted bounding box and the ground-truth bounding box exceeds 50%; otherwise, it is considered a false positive. We also report the *mean-average precision* (mAP) which is the mean of AP across all categories. We invite the reader to refer to the Pascal VOC development kit document [9] for more details on the AP

9. http://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf

metric.

The main difference between our localization challenge and other localization challenges is the occurrence of the *Motorized Vehicle* category. This category is used to label vehicles that are too small (or too occluded) to be correctly assigned a specific category. For example, a car seen from far away and whose size does not exceed a few pixels would be labeled as a *Motorized Vehicle*. In order not to penalize methods which would incorrectly label those small objects, every *Motorized Vehicle* labeled by one of following categories: *Articulated Truck, Bus, Car, Pickup Truck, Single-Unit Truck* and *Work Van*, is considered a true detection. In our implementation, we first identify every predicted bounding box whose overlap ratio with a *Motorized Vehicle*'s ground-truth bounding box exceeds 50%, and re-assign its category label to *Motorized Vehicle*. Then, we compute the AP for each category.

A.4 Methods Tested

One of the overarching objectives of this paper is to identify how far state-of-the-art machine learning methods can go at classifying and localizing vehicles pictured by real traffic surveillance cameras. As mentioned earlier, this implies the analysis of low resolution 2D images with strong compression artifacts, recorded during daytime/nighttime, in different seasons, under diverse weather conditions, and with various camera positions and orientations. As such, one can expect that some methods might be robust to those challenges while others may not. In order to identify solved and unsolved issues, we carefully benchmarked a series of state-of-the-art deep learning methods. In this section, we describe the methods that we tested on the MIO-TCD classification and localization datasets. Some methods have been published before the release of the MIO-TCD dataset while others have been designed specifically for this dataset's challenges [63].

A.4.1 Vehicle Classification

Pre-Trained CNN Features + SVM

The first series of methods aim at gauging how "different" is the statistical content of traffic images in the MIO-TCD dataset from other datasets, such as ImageNet. We did so

by training a linear SVM classifier on CNN features obtained from ImageNet pre-trained CNN models. This is inspired by Razavian *et al.* [87] who demonstrated that features obtained from deep CNN models could be the primary choice of a large variety of visual recognition tasks. We did so with the following six pre-trained deep models: AlexNet [50], InceptionV3 [90], ResNet-50 [36], VGG-19 [89], Xception [10] and DenseNet [39]. The layer we used to extract the features from is: ReLU of FC-7 in AlexNet and VGG-19, and the global pooling layer in InceptionV3, ResNet-50, Xception and DenseNet. Their corresponding feature dimensions are 4096, 4096, 2048, 2048, 2048, 1920. The python *scikit-learn* library [10] was used to train the linear SVM with $C = 1$.

Retrained CNN Models

Although features from the first layers of a CNN model are independent from the dataset it was trained on (mostly Gabor-like filters [99]), we retrained end-to-end all six CNN models on our classification dataset. Those models were all initialized with ImageNet pre-trained weights and were trained with the loss function proposed in the corresponding original papers. We used the Adam [47] optimizer with a learning rate of 10^{-3} that we empirically found more effective than other optimizers such as RMSprop or SGD. Training was done for a maximum of 50 epochs with a validation-based early stopping criteria with a patience of 10 epochs to prevent over-fitting. The batch size was adjusted to each model so it could fit on our 12GB Titan X GPUs. Please note that we included a series of batch normalization layers [41] in AlexNet and VGG-19 to speed up training. All models but DenseNet were implemented with the Keras library [9]. DenseNet was implemented with PyTorch [75].

We also implemented the following four training configurations to further improve results on our dataset:

— The first configuration is a basic training with normal sampling of the data.

— The second configuration involves data augmentation using horizontal flipping and randomized shearing and zooming to enrich the training dataset.

— Since the dataset is highly unbalanced (see Table A.1), we used data augmentation with uniform sampling. That is, at each epoch we used an equal number of images

10. scikit-learn.org/

from each class to prevent large classes, such as *Car*, *Pickup Truck* and *Background* from gaining too much importance over smaller classes, such as *Motorcycle*.

— As suggested by Havaei *et al.* [33], we implemented a two-phase training procedure. In the first phase, we used data augmentation with uniform sampling. In the second phase, we froze the entire network except for the last layer which was retrained with data augmentation and normal sampling.

MIO-TCD Classification (Ensemble Models)

In the wake of the 2017 CVPR MIO-TCD Challenge [63], several methods have been designed for the sole purpose of classifying traffic images. Interestingly, all of those methods involve a combination of several deep learning models. Kim and Lim [46] proposed a bagging system where several CNN models are trained on a random subset of the MIO-TCD dataset. Their final result is obtained with a weighted majority vote to compensate for the unbalanced nature of the dataset. Lee and Chung [52] proposed an ensemble method which combines 3 convolutional nets (AlexNet, GoogleNet and ResNet18) trained on 18 different sets of data. GoogleNet was trained on 12 subsets of the dataset (aka the local nets) and AlexNet, GoogleNet and ResNet18 were trained on the entire dataset but with different image sizes (aka the global nets). At the test time, the networks are selected with a gating function and combined with a *softmax* layer. Jung *et al.* [44] proposed an ensemble model according to which several deep residual networks are jointly trained. The main novelty of their method lies within its loss function which allows to train every ResNet simultaneously. Theagarajan *et al.* [91] also proposed an ensemble of ResNet models. The authors implemented a weighted loss function to account for the unbalanced nature of the dataset. They also implemented a patch-wise logical reasoning process to disambiguate classes that are close to each other like trucks and buses.

A.4.2 Vehicle Localization

Recent CNN-Based Methods

Recently, CNN-based localization methods established state-of-the-art performance on several object localization datasets. In this paper, we evaluated Faster R-CNN [82], SSD-300, SSD-512 [56], YOLO [80] and YOLO-v2 [79].

A.4. METHODS TESTED

Faster R-CNN is an improved version of Girshich *et al.*'s R-CNN [23] and Fast R-CNN [22] methods. Unlike R-CNN and Fast R-CNN, that use a selective search [92] to generate object bounding box proposals, Faster R-CNN has a region proposal network that can directly estimate proposals based on the CNN feature maps thus making it end-to-end trainable. Liu *et al.* [56] improved upon the Faster R-CNN model with their SSD (Single Shot MultiBox Detector) framework which generates object proposals with feature maps from multiple layers. As for YOLO (You Only Look Once) [80], it takes a 448×448 image as input and outputs object detections within a 7×7 grid. Later on, Redmon *et al.* [79] integrated anchor boxes (for computing potential bounding boxes) into their YOLO model. We refer to this model as YOLO-v2.

Similarly to the comparison of pre-trained CNN features and retrained CNN models for classification, we trained all localization models with and without updating the weights inherited from an ImageNet pre-trained model to observe the efficiency of ImageNet features on vehicle localization (i.e. we only trained the layers that weren't initialized with the ImageNet weights). We trained Faster R-CNN end-to-end for 300,000 iterations with code provided by the authors [11]. As recommended by the original paper, we used VGG-16 as a pre-trained model. Similarly, for SSD-300 and SSD-512 (i.e., SSD with input images resized to 300×300 and 512×512, respectively) we used the code released by the authors [12]. The SSD-300 model was trained for 120,000 iterations with a batch-size of 32, and the SSD-512 model was trained for 240,000 iterations with a batch-size of 16. YOLO was trained using the Darknet deep learning toolbox [13]. Both YOLO and YOLO-v2 were trained for 80,000 iterations with a batch-size of 64. As YOLO-v2 needs pre-clustered anchor bounding boxes, we evaluated two kinds of anchor boxes: boxes computed on the Pascal VOC datasets (referred to as YOLO-v2(P)) and on our localization dataset (referred to as YOLO-v2(M)).

MIO-TCD Localization

We report results from two localization methods designed specifically for the MIO-TCD dataset. The first one is from Wang *et al.* [95] which improves methods such as Faster

11. https://github.com/rbgirshick/py-faster-rcnn
12. https://github.com/weiliu89/caffe/tree/ssd
13. https://pjreddie.com/darknet/

R-CNN and SSD by leveraging the scene context. The idea is to combine the *softmax* score of these methods with a context term which the authors model with a k-NN algorithm. The second method is from Jung *et al.* [44] which combines the results computed from multiple R-FCN models [12] trained with different backbones (ResNet-50 and ResNet-101) together with a joint non-maximal suppression step to localize vehicles.

A.5 Experimental Results

In this section, we report experimental results obtained on the MIO-TCD classification and localization datasets with the deep learning methods presented earlier.

A.5.1 Vehicle Classification

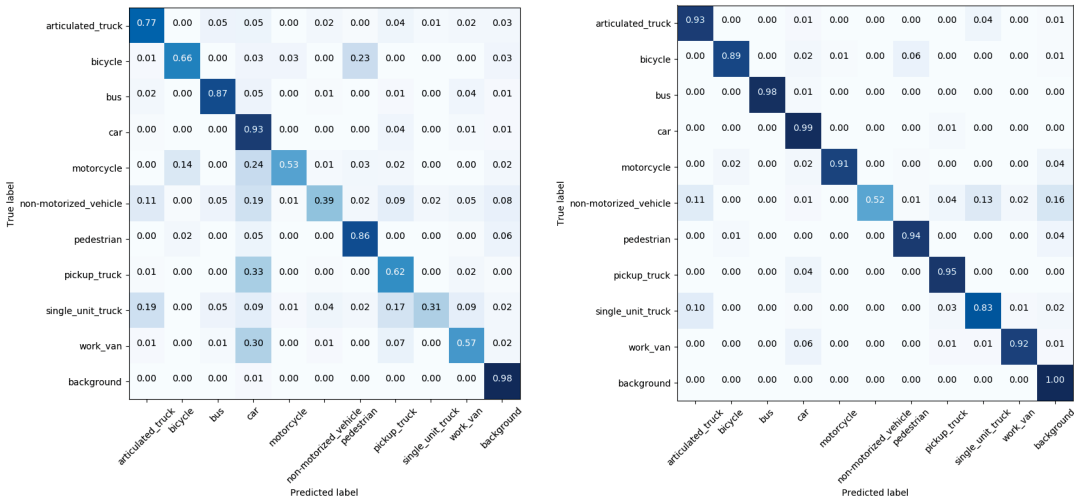


Figure A.3 – Confusion matrices obtained on the classification dataset with pre-trained ResNet-50 features + SVM on left and the ensemble model by Jung *et al* [44] on right.

Pre-Trained CNN Features + SVM

Results obtained with SVM trained on features extracted from six pre-trained CNN models are shown in Table A.2. All six models have an accuracy of more than 80%, which

Table A.2 – Evaluation metrics for six pre-trained models used with linear SVM classifiers on the classification dataset.

	Acc	mRe	mPr	$Kappa$
AlexNet	0.82	0.49	0.55	0.72
Inception-V3	0.84	0.57	0.64	0.75
ResNet-50	**0.89**	**0.69**	0.74	**0.83**
VGG19	0.83	0.66	0.59	0.75
Xception	0.87	0.54	0.76	0.78
DenseNet	0.86	0.51	**0.82**	0.78

is surprisingly high considering the different nature of ImageNet and MIO-TCD datasets. However, careful analysis reveals that these methods have relatively low mean recall, mean precision and Kappa score. To understand this situation, one has to consider the confusion matrix of ResNet-50 shown in Fig. A.3. While the *Car* and *Background* categories have an accuracy of more than 90%, others, such as *Non-Motorized Vehicle*, *Motorcycle* and *Single-Unit Truck* categories, suffer from very low accuracy.

These numbers can be explained by the unbalanced nature of the dataset, as the *Car* and *Background* categories contain more than 80% of all images. In this case, large classes strongly influence the decision boundaries to the detriment of smaller classes. This also explains why several categories have a large proportion of samples wrongly classified as *Car*. It is the case for *Work Van* for which 30% of its images are classified as *Car*. A detailed analysis revealed that features trained on ImageNet do not discriminate family vans (labeled as *Car* in the dataset) from *Work Vans*. Confusion also happens between *Bicycle* and *Pedestrian*, as well as within the group of *Articulated Truck*, *Pickup Truck* and *Single-Unit Truck*.

Based on these results, we conclude that SVM trained on ImageNet CNN features is accurate at classifying large classes but this does not generalize well to smaller classes.

Retrained CNN Models

Table A.3 reports evaluation metrics for the six models trained in four different training configurations. As can been seen, there is a substantial performance increase in comparison to the SVM results from Table A.2. Results show that data augmentation improves the

Table A.3 – Evaluation metrics for retrained CNN models on the classification dataset in four different configurations. 'N' stands for normal sampling, 'U' for uniform sampling, 'D' is for using data augmentation and 'T' is a two-phase training procedure.

	Acc				mRe				mPr				$Kappa$			
	N	D+N	D+U	D+U(T)	N	D+N	D+U	D+U(T)	N	D+N	D+U	D+U(T)	N	D+N	D+U	D+U(T)
AlexNet	0.869	0.892	0.820	0.883	0.631	0.560	0.826	0.417	0.546	0.692	0.631	0.899	0.796	0.832	0.743	0.807
Inception-V3	0.947	0.962	0.929	0.959	0.809	0.828	0.888	0.872	0.792	0.848	0.763	0.889	0.918	0.941	0.892	0.937
ResNet-50	0.938	0.967	**0.959**	**0.969**	0.795	0.871	0.892	**0.877**	0.734	0.858	0.865	0.903	0.902	0.948	**0.936**	**0.952**
VGG-19	0.941	0.956	0.887	0.940	0.773	0.824	0.654	0.817	0.794	0.853	0.838	0.841	0.909	0.932	0.831	0.907
Xception	0.958	**0.976**	0.941	0.961	0.846	**0.908**	0.822	0.819	0.794	0.906	**0.900**	0.836	0.935	**0.963**	0.910	0.939
DenseNet	**0.970**	0.973	0.952	0.954	**0.889**	0.870	**0.896**	0.825	**0.886**	**0.916**	0.850	**0.908**	**0.953**	0.958	0.927	0.929

Kappa score of every model, especially ResNet-50. Note that although data augmentation may reduce the mean recall of a certain method, this is compensated by a larger increase of the mean precision. However, the use of uniform sampling (+U) with and without the two-phase training procedure (T) does not improve results over the normal sampling (+N).

A careful inspection reveals that uniform sampling has a positive impact on small categories (such as *Motorcycle*, for example), but decreases the performance for large categories.

Overall, Xception and DenseNet with data augmentation and normal sampling are the best methods with very similar performance. VGG-19, ResNet-50 and Inception-V3 also get very accurate results with Kappa scores above 0.93.

MIO-TCD Classification (Ensemble Models)

Table A.4 reports results submitted to the 2017 CVPR MIO-TCD Challenge. These are ensemble methods which combine the outputs of different models. As can be seen, all these methods have similar performance with accuracy of about 0.98 and Kappa score of almost 0.97. With these results being only marginally better than those obtained with Xception and DenseNet, we conclude that the combination of several models does not bring much for a dataset such as MIO-TCD.

Error Analysis

Results from Tables A.3 and A.4 reveal that despite large illumination variations between images, compression artifacts, arbitrary vehicle orientation, poor resolution and inter-class similarities, the classification of traffic vehicles seems almost solved. However, in-depth analysis of the top-performing methods reveals some unsolved issues. In Fig. A.3,

Table A.4 – Evaluation metrics for ensemble models from the 2017 MIO-TCD Classification challenge.

	Acc	*mRe*	*mPr*	*Kappa*
Kim and Lim [46]	0.9786	0.9041	0.9355	0.9666
Lee and chung [52]	0.9792	0.9024	0.9298	0.9675
Jung *et al* [44]	**0.9795**	0.8970	**0.9530**	**0.9681**
Theagarajan *et al* [91]	0.9780	**0.9190**	0.9439	0.9658

we show the confusion matrix for the method by Jung *et al.* [44] whose performance is globally similar to that obtained by other top performing methods. As one can see, *Non-Motorized Vehicles* are poorly handled. Images of *Non-Motorized Vehicles* in our dataset include a wide variety of trailers pulled by a vehicle, typically a car or a pickup truck. As shown in Fig. A.4 (second row, third column), *Non-Motorized Vehicles* are often wrongly classified as a single-unit or an articulated truck, as their shapes are very similar.

Without much surprise, methods also get confused between categories with similar visual characteristics, such as the *Work Van* class and the *Car* class (more specifically, family vans considered to belong to the *Car* class) or the *Articulated Truck* and the *Single-Unit Truck*. They also get confused with vehicles that look unusual. For example, in Fig. A.4, the blue car with a black top (second row, first column) gets wrongly classified as a pickup truck and the pickup truck with a cap (third row, fist column) is wrongly classified as a car. Also, classes with small objects such as *Pedestrian*, *Bicycle* and *Motorcycle* often suffer from heavy compression artifacts and are thus more likely to be mis-classified.

A.5.2 Vehicle Localization

Recent CNN-Based Methods

The average precision of localization for Faster R-CNN, SSD-300, SSD-512, YOLO-v1, YOLO-v2(P) and YOLO-v2(M) methods is shown in Table A.5.

Methods followed by "(w/o)" were trained without updating the weights inherited from a pre-trained ImageNet model. Note that when training Faster R-CNN with frozen layers, we found that it failed to converge. This behavior (also reported in [88]) is mainly due to gradient issues when using RoI pooling layers. For the other models, we see that training them in full is highly beneficial, with a very significant boost of mAP (24.1–42.5%

Figure A.4 – Examples of failure cases from top-performing methods for every class where the yellow label (top) is the ground truth and the white label (bottom) is the predicted class.

Table A.5 – Average precision (AP) of localization for Faster R-CNN, SSD, YOLO and two methods submitted to the MIO-TCD Challenge on localization. ("w/o": without updating the weights inherited from an ImageNet pre-trained model)

	mAP	Articulated Truck	Bicycle	Bus	Car	Motorcycle	Motorized Vehicle	Non-Motorized Vehicle	Pedestrian	Pickup Truck	Single-Unit Truck	Work Van
Faster R-CNN (w/o)	6.3	4.5	2.0	3.0	5.5	3.7	6.6	0.9	0.4	10.0	2.4	2.6
SSD-300 (w/o)	36.4	44.1	52.2	68.0	55.0	38.5	29.4	3.2	10.7	55.6	19.3	24.7
SSD-512 (w/o)	34.8	45.2	25.6	71.0	61.9	30.0	37.0	0.8	12.2	61.0	15.8	22.0
YOLO-v1 (w/o)	31.3	48.2	19.2	78.5	50.6	15.4	17.3	6.3	2.3	61.8	12.7	31.9
YOLO-v2(P) (w/o)	46.3	54.6	53.5	82.8	61.4	56.8	26.8	20.5	9.9	68.8	30.2	43.5
YOLO-v2(M) (w/o)	47.7	60.9	54.2	85.9	62.0	60.7	27.1	19.2	8.8	69.9	32.3	43.7
Faster R-CNN	70.0	85.9	78.4	95.2	82.6	81.1	52.8	37.3	31.3	89.0	62.5	73.6
SSD-300	74.0	90.6	78.3	95.7	91.5	78.9	51.4	55.2	37.3	90.7	69.0	75.0
SSD-512	**77.3**	**92.1**	**78.6**	**96.8**	**94.0**	**82.3**	**56.8**	**58.8**	**43.6**	**93.1**	**74.0**	**80.4**
YOLO-v1	62.7	82.7	70.0	91.6	77.2	71.4	44.4	20.7	18.1	85.6	58.3	69.3
YOLO-v2(P)	71.5	86.7	78.4	95.2	80.5	80.9	52.0	56.5	25.7	84.6	70.0	75.7
YOLO-v2(M)	71.8	88.3	**78.6**	95.1	81.4	81.4	51.7	56.6	25.0	86.5	69.2	76.4
Wang *et al.*[95]	77.2	91.6	79.9	96.8	**93.8**	83.6	56.4	58.2	42.6	**92.8**	73.8	79.6
Jung *et al.*[44]	**79.2**	**92.5**	**87.3**	**97.5**	89.7	**88.2**	**62.3**	**59.1**	**48.6**	92.3	**74.4**	**79.9**

improvement).

From this point, we only discuss results obtained with fully-trained models. As can be seen, the SSD methods outperform both Faster R-CNN and YOLO, with SSD-512 being the best-performing method with a mAP of 77.3%.

Results for SSD-300 and SSD-512 show that increasing input image resolution from 300×300 to 512×512 improves the mAP by 4%. Also, YOLO-v2 has the mAP 10% higher than YOLO-v1 thus showing that anchor boxes are useful features. Furthermore, results for YOLO-v2(P) and YOLO-v2(M) show that anchor boxes computed on our localization dataset marginally improve mAP over anchor boxes pre-computed from the Pascal VOC dataset.

Here again, the largest classes, namely *Car*, *Pickup Truck*, *Articulated Truck* and *Bus* get the best results with an average precision above 80% for almost every method, while *Motorized Vehicle*, *Non-Motorized Vehicle* and *Pedestrian* are the three categories with the lowest average precision. The main challenge with *Motorized Vehicle* and *Pedestrian* classes stems from the small size of vehicles that are likely to be confused with the *Bicycle* and *Motorcycle* categories. As for the *Non-Motorized Vehicle*, similarly to the classification dataset, it is often confused with the *Articulated Truck* and *Single-Unit Truck* classes.

In Fig. A.5, we show some detection results for different methods. While most methods can accurately localize large and well-contrasted vehicles, we can see that Faster R-CNN is prone to false detections while YOLO-v1 and YOLO-v2 suffer from mis-detections of small objects (typically, pedestrians).

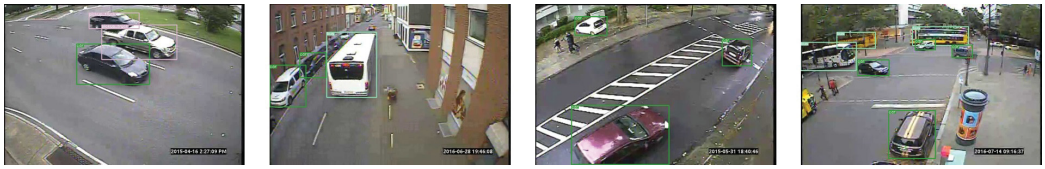
MIO-TCD Localization

At the bottom of Table A.5, are shown localization metrics for methods by Wang *et al.* [95] and Jung *et al.* [44] submitted to the MIO-TCD Challenge. Both methods attain excellent performance, with the Jung *et al.* method achieving the best mAP of 79.2% and outperforming state-of-the-art CNNs for almost all vehicle classes.

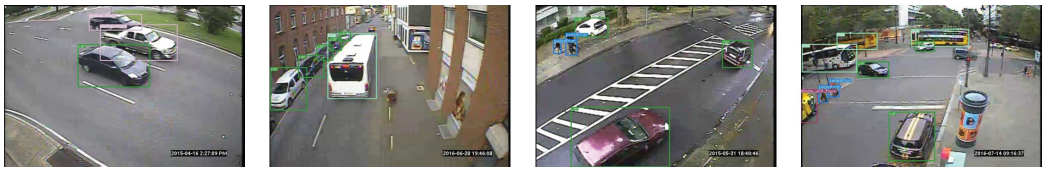
Detailed Analysis

We now thoroughly analyze the influence of object scale on the performance of localization methods as well as the nature of false detections. We do so *via* the Microsoft

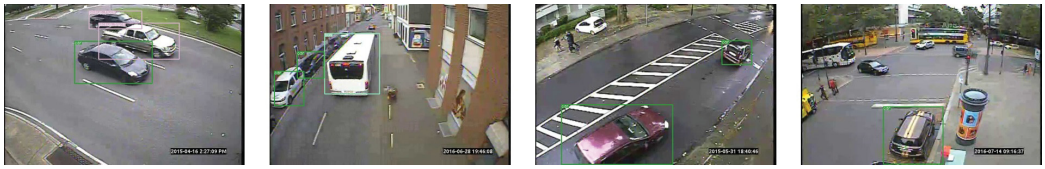
(a) Faster R-CNN



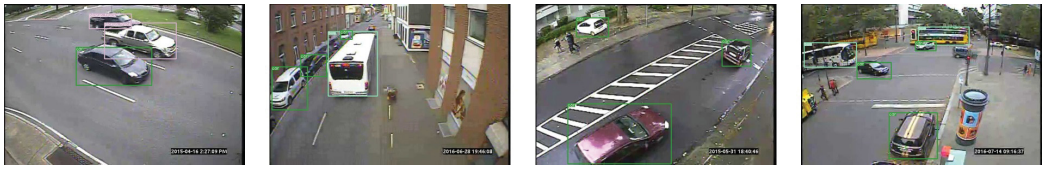
(b) SSD-300



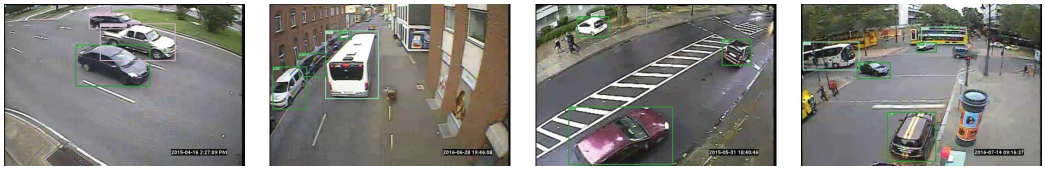
(c) SSD-512



(d) YOLO-v1



(e) YOLO-v2 (Pascal VOC)



(f) YOLO-v2 (MIO-TCD)

Figure A.5 – Detection examples on the localization dataset for Faster R-CNN, SSD-300, SSD-512, YOLO, YOLO-v2 (Pascal VOC) and YOLO-v2 (MIO-TCD). We only show detections with probability scores higher than 0.6.

COCO's evaluation procedure [53] and the object detectors' protocol by Hoeim *et al.* [38].

Scale Every object has been classified as belonging to one of 3 scales: small objects with bounding box area below 32^2, medium objects with the area between 32^2 and 96^2, and large objects with the area larger than 96^2. The average precision for each of these scales is reported in Table. A.6. As can be seen, all methods in the table are ill-suited for detecting small objects, in our case *Pedestrian*, *Bicycle*, *Motorcycle* classes as well as vehicles seen from a distance (see Fig. A.2 for examples of small vehicles due to perspective). Furthermore, when increasing the overlap ratio for correct detections from 0.5 to 0.75, we find that the AP of Faster R-CNN and YOLO decreases by almost 30%, while for SSD it decreases by around 15%. This means that the bounding boxes estimated by the SSD method are tighter around the ground-truth bounding boxes.

Table A.6 – Average precision of localization computed using Microsoft COCO's evaluation protocol.

	Average Precision				
	Overlap		Scale		
	0.5	0.75	small	medium	large
Faster R-CNN	70.0	38.5	14.3	40.2	55.1
SSD-300	74.3	57.1	21.5	53.3	69.0
SSD-512	**77.6**	**61.9**	**28.2**	**57.3**	**72.5**
YOLO-v1	62.6	34.0	11.3	32.4	52.7
YOLO-v2(P)	71.3	42.7	15.7	41.3	59.3
YOLO-v2(M)	71.8	43.0	16.0	41.7	61.3
Wang *et al.* [95]	77.4	**59.9**	**26.6**	**55.6**	60.7
Jung *et al.* [44]	**79.3**	58.8	26.5	54.9	**69.3**

False positives To examine the nature of false positives, we follow the methodology of Hoiem *et al.* [38] according to which each prediction is either correct or wrongly classified into one of the following errors:

— **Localization**: the predicted bounding box has a correct label but is misaligned with the ground-truth bounding box ($0.1 <$ Overlap < 0.5).

— **Similarity**: the predicted bounding box has Overlap > 0.1 with a ground-truth bounding box but its predicted label is incorrect. However, the predicted label be-

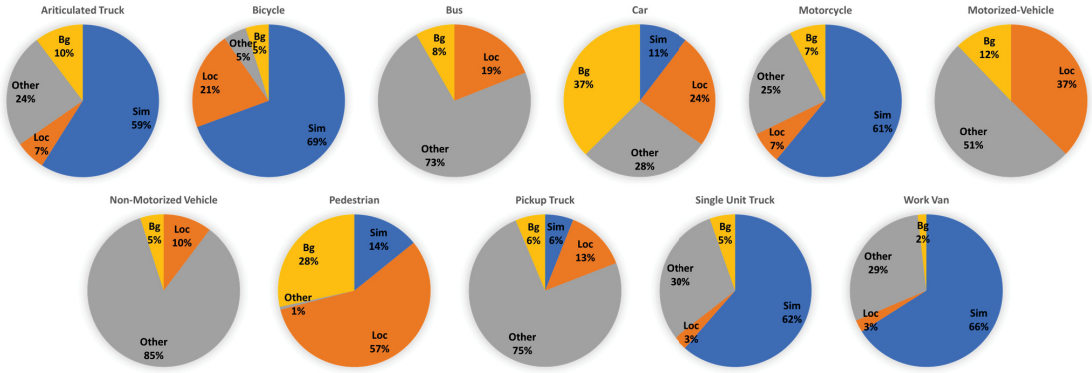


Figure A.6 – Analysis of false detections for the SSD-300 method. Each pie-chart shows the fraction of top-ranked false positives of each category due to poor localization (Loc), confusion with similar categories (Sim), confusion with other categories (Other), or confusion with background or unlabeled objects (Bg).

longs to one of three similarity classes (groups of similar classes), namely: {*Articulated Truck, Pickup Truck, Single-Unit Truck*}, {*Bicycle, Motorcycle, Pedestrian*}, and {*Car, Work Van*}.

— **Other**: the predicted bounding box has a Overlap > 0.1 with a ground-truth bounding box but its predicted label is incorrect and does not fall within a group of similar classes.

— **Background**: all other false positives are classified as background, mainly confused with unlabeled objects.

Fig. A.6 shows the frequency of occurrence of each type of error for the SSD-300 method. For the similarity class {*Articulated Truck, Pickup Truck, Single-Unit Truck*}, the *Articulated Truck* and *Single-Unit Truck* classes are likely to be confused with each other, while the *Pickup Truck* is confused with other classes, mostly *Car*. As for the {*Car, Work Van*} similarity class, we find that 66% of *Work Van* false positives are wrongly classified as *Car* for the reason mentioned before (*Work Van* is often confused with a family van). As for *Car* false detections, the confusion is mostly with *Background*. For the {*Bicycle, Motorcycle, Pedestrian*} similarity class, the *Bicycle* and *Motorcycle* classes are often confused with each other, while *Pedestrian*, due to small scale, suffers from localization errors. As for the *Bus*, *Motorized Vehicle* and *Non-Motorized Vehicle* classes, they all suffer from confusion with other categories.

A.6 Discussion and Conclusions

In this paper, we introduced the *MIOvision Traffic Camera Dataset (MIO-TCD)*, the largest dataset to date for motorized traffic analysis. The dataset consists of two parts: a "localization dataset", containing full video frames with bounding boxes around traffic objects, and a "classification dataset", containing crops of 11 types of traffic objects.

We evaluated several state-of-the-art deep learning methods on the MIO-TCD dataset. Results show that well-trained models reach impressive accuracy and Kappa scores of more than 96% on the classification dataset and a mean-average precision of 79% on the localization dataset.

While Xception and DenseNet with data augmentation are the best classification methods, VGG-19, RestNet-50 and Inception-V3 attain very good results as well. As for the ensemble models, they reach, for all practical purposes, the same scores as Xception and DenseNet. A careful inspection of results reveals that *Non-Motorized Vehicles* is the only problematic class with a precision below 80%. Other errors in the top results can be explained by the confusion between classes with similar visual characteristics such as *Single-Unit Truck* and *Articulated Truck*.

As for localization methods, the method by Jung *et al.* [44] gets the best scores (mAP = 79.2%) but is closely followed by SSD-512 (mAP = 77.3%). A detailed analysis reveals that errors often happen between similar classes or are due to a mis-alignment of the predicted bounding box (overlapping ratio below 0.5).

In light of these results, we may conclude that state-of-the-art deep learning methods exhibit a capacity to localize and recognize vehicles from single video frames without the need for dynamic features captured by video, as was required to date. This opens the door to new, low-frame-rate video analytics applications such as traffic statistics, traffic density estimation, car counting, and anomaly detection.

Although deep models achieve very promising performance, a number of challenges remain, among them: similarly-looking vehicles, unbalanced data, false detections and small vehicles. We conclude the paper by discussing these challenges below.

1. *Similarly-looking vehicles*: To differentiate vehicles with similar appearance, such as bicycles and motorcycles, it may be necessary to identify semantic features in addition to global features.

2. *Unbalanced data*: Experiments show that ensemble models can deal with this issue to some extent, at the cost of significantly more computations. It would be beneficial to develop algorithms that leverage the benefits of ensemble models at reduced computational load.
3. *False detections*: The appearance and position of different vehicles is highly correlated with scene layout, such as road direction. It would be preferable to embed the layout information into the localization model as a means of enhancing detection and reducing false positives.
4. *Small vehicles*: The localization results indicate that large vehicles are more easily detected than small vehicles. Adversarial training or metric learning methods projecting vehicles of various sizes into the same feature space could be an avenue towards improving the localization and classification accuracy of small vehicles.

Bibliographie

[1] David P. Wipf BIN DAI, Chen Zhu.
« Compressing Neural Networks using the Variational Information Bottleneck ».
proc. of ICML, 2018.

[2] Christopher M. BISHOP.
Pattern Recognition and Machine Learning (Information Science and Statistics).
Springer-Verlag, 2006.

[3] Stephen BOYD, Neal PARIKH, Eric CHU, Borja PELEATO, Jonathan ECKSTEIN et OTHERS.
« Distributed optimization and statistical learning via the alternating direction method of multipliers ».
Foundations and Trends® in Machine learning, 3(1):1–122, 2011.

[4] Stephen BOYD et Lieven VANDENBERGHE.
Convex Optimization.
Cambridge University Press, 2004.

[5] Andrew BROCK, Jeff DONAHUE et Karen SIMONYAN.
« Large scale gan training for high fidelity natural image synthesis ».
arXiv preprint arXiv :1809.11096, 2018.

[6] Miguel CARREIRA-PERPINAN et Weiran WANG.
« Distributed optimization of deeply nested systems ».
Dans *Int. Conf. Artificial Intelligence and Statistics*, pages 10–19, 2014.

[7] Miguel A CARREIRA-PERPINÁN et Yerlan IDELBAYEV.
« Learning-Compression Algorithms for Neural Net Pruning ».
Dans *Proc. of CVPR*, pages 8532–8541, 2018.

[8] Guobin CHEN, Wongun CHOI, Xiang YU, Tony HAN et Manmohan CHANDRAKER.
« Learning efficient object detection models with knowledge distillation ».
Dans *Advances in Neural Information Processing Systems*, pages 742–751, 2017.

[9] François CHOLLET et OTHERS.
« Keras », 2015.
https://github.com/fchollet/keras.

[10] François CHOLLET.
« Xception : Deep Learning with Depthwise Separable Convolutions ».
Dans *Proc. CVPR*, pages 1800–1807, 2017.

[11] Marius CORDTS, Mohamed OMRAN, Sebastian RAMOS, Timo REHFELD, Markus ENZWEILER, Rodrigo BENENSON, Uwe FRANKE, Stefan ROTH et Bernt SCHIELE.
« The cityscapes dataset for semantic urban scene understanding ».
Dans *Proc.CVPR*, pages 3213–3223, 2016.

[12] Jifeng DAI, Yi LI, Kaiming HE et Jian SUN.
« R-FCN : Object detection via region-based fully convolutional networks ».
Dans *Proc. NIPS*, pages 379–387, 2016.

[13] Navneet DALAL et Bill TRIGGS.
« Histograms of oriented gradients for human detection ».
Dans *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[14] Navneet DALAL et Bill TRIGGS.
« Histograms of oriented gradients for human detection ».
Dans *Proc. CVPR*, pages 886–893, 2005.

[15] Misha DENIL, Babak SHAKIBI, Laurent DINH, Nando DE FREITAS et OTHERS.
« Predicting parameters in deep learning ».
Dans *proc of NIPS*, pages 2148–2156, 2013.

[16] Emily L DENTON, Wojciech ZAREMBA, Joan BRUNA, Yann LECUN et Rob FERGUS.
« Exploiting linear structure within convolutional networks for efficient evaluation ».
Dans *proc of NIPS*, pages 1269–1277, 2014.

[17] Piotr DOLLAR, Christian WOJEK, Bernt SCHIELE et Pietro PERONA.
« Pedestrian detection : An evaluation of the state of the art ».
IEEE Trans. Pattern Anal. Mach. Intell., 34(4):743–761, 2012.

[18] Zhen DONG, Yuwei WU, Mingtao PEI et Yunde JIA.
« Vehicle type classification using a semisupervised convolutional neural network ».
IEEE Trans. Int. Trans. Sys., 16(4):2247–2256, 2015.

[19] Mark EVERINGHAM, Luc VAN GOOL, Christopher KI WILLIAMS, John WINN et Andrew ZISSERMAN.
« The pascal visual object classes (voc) challenge ».
Int. J. Comput. Vision, 88(2):303–338, 2010.

[20] Jonathan FRANKLE et Michael CARBIN.
« The lottery ticket hypothesis : Training pruned neural networks ».
arXiv preprint arXiv :1803.03635, 2018.

[21] Andreas GEIGER, Philip LENZ et Raquel URTASUN.
« Are we ready for autonomous driving ? the kitti vision benchmark suite ».
Dans *Proc. CVPR*, pages 3354–3361, 2012.

[22] Ross GIRSHICK.
« Fast r-cnn ».
Dans *Proc. ICCV*, pages 1440–1448, 2015.

[23] Ross GIRSHICK, Jeff DONAHUE, Trevor DARRELL et Jitendra MALIK.
« Rich feature hierarchies for accurate object detection and semantic segmentation ».
Dans *Proc. CVPR*, pages 580–587, 2014.

[24] Yunchao GONG, Liu LIU, Ming YANG et Lubomir BOURDEV.
« Compressing deep convolutional networks using vector quantization ».
arXiv preprint arXiv :1412.6115, 2014.

[25] Ian GOODFELLOW, Yoshua BENGIO et Aaron COURVILLE.
Deep Learning.
MIT Press, 2016.
http://www.deeplearningbook.org.

[26] A. GORDON, E. EBAN, O. NACHUM, B. CHEN, H. WU, T. YANG et E. CHOI.
« MorphNet : Fast amp ; Simple Resource-Constrained Structure Learning of Deep

Networks ».
Dans *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1586–1595, June 2018.

[27] Ricardo GUERRERO-GÓMEZ-OLMEDO, Roberto J LÓPEZ-SASTRE, Saturnino MALDONADO-BASCÓN et Antonio FERNÁNDEZ-CABALLERO.
« Vehicle tracking by simultaneous detection and viewpoint estimation ».
Dans *Proc. WCIBNAC*, pages 306–316, 2013.

[28] Ricardo GUERRERO-GÓMEZ-OLMEDO, Beatriz TORRE-JIMÉNEZ, Roberto LÓPEZ-SASTRE, Saturnino MALDONADO-BASCÓN et Daniel OÑORO-RUBIO.
« Extremely overlapping vehicle counting ».
Dans *in proc of ICPRIA*, pages 423–431, 2015.

[29] Emil Julius GUMBEL.
« Les valeurs extrêmes des distributions statistiques ».
Annales de l'institut Henri Poincaré, 5(2):115–158, 1935.

[30] Song HAN, Huizi MAO et William J DALLY.
« Deep Compression : Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding ».
proc. of ICLR, 2016.

[31] Song HAN, Jeff POOL, John TRAN et William DALLY.
« Learning both weights and connections for efficient neural network ».
Dans *proc of NIPS*, pages 1135–1143, 2015.

[32] Babak HASSIBI et David G STORK.
« Second order derivatives for network pruning : Optimal brain surgeon ».
Dans *proc of NIPS*, pages 164–171, 1993.

[33] M. HAVAEI, A. DAVY, D. WARDE-FARLEY, A. BIARD, A. COURVILLE, Y. BENGIO, C. PAL, P-M. JODOIN et H. LAROCHELLE.
« Brain Tumor Segmentation with Deep Neural Networks ».
Med. Image Anal., 35:18–31, 2017.

[34] Kaiming HE, Georgia GKIOXARI, Piotr DOLLÁR et Ross GIRSHICK.
« Mask r-cnn ».
Dans *proc. of ICCV*, pages 2980–2988, 2017.

[35] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN.
« Deep Residual Learning for Image Recognition ».
Dans *Proc. of CVPR*, pages 770–778, June 2016.

[36] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN.
« Deep residual learning for image recognition ».
Dans *Proc. CVPR*, pages 770–778, 2016.

[37] Geoffrey HINTON, Oriol VINYALS et Jeff DEAN.
« Distilling the knowledge in a neural network ».
proc of NIPS DLRL Workshop, 2015.

[38] Derek HOIEM, Yodsawalai CHODPATHUMWAN et Qieyun DAI.
« Diagnosing error in object detectors ».
Proc. ECCV, 2012.

[39] G. HUANG, Z. LIU, L. v. d. MAATEN et K. Q. WEINBERGER.
« Densely Connected Convolutional Networks ».
Dans *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, July 2017.

[40] Forrest N IANDOLA, Song HAN, Matthew W MOSKEWICZ, Khalid ASHRAF, William J DALLY et Kurt KEUTZER.
« Squeezenet : Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size ».
arXiv preprint arXiv :1602.07360, 2016.

[41] Sergey IOFFE et Christian SZEGEDY.
« Batch normalization : Accelerating deep network training by reducing internal covariate shift ».
arXiv preprint arXiv :1502.03167, 2015.

[42] Max JADERBERG, Andrea VEDALDI et Andrew ZISSERMAN.
« Speeding up convolutional neural networks with low rank expansions ».
proc of BMVC, 2014.

[43] J.COHEN.
« A coefficient of agreement for nominal scales ».
Educ. Psychol. Meas., 20(1):37–46, 1960.

[44] Heechul JUNG, Min-Kook CHOI, Jihun JUNG, Jin-Hee LEE, Soon KWON et Woo Young JUNG.
« ResNet-Based Vehicle Classification and Localization in Traffic Surveillance Systems ».
Dans *Proc. CVPRW*, pages 934–940, 2017.

[45] V KASTRINAKI, Michalis ZERVAKIS et Kostas KALAITZAKIS.
« A survey of video processing techniques for traffic applications ».
Image vision comput., 21(4):359–381, 2003.

[46] Pyong-Kun KIM et Kil-Taek LIM.
« Vehicle Type Classification Using Bagging and Convolutional Neural Network on Multi View Surveillance Image ».
Dans *Proc. CVPRW*, pages 914–919, 2017.

[47] D. KINGMA et J. BA.
« Adam : A Method for Stochastic Optimization ».
Dans *Proc. ICLR*, 2015.

[48] Diederik P KINGMA, Tim SALIMANS et Max WELLING.
« Variational dropout and the local reparameterization trick ».
Dans *proc of NIPS*, pages 2575–2583, 2015.

[49] Jonathan KRAUSE, Michael STARK, Jia DENG et Li FEI-FEI.
« 3d object representations for fine-grained categorization ».
Dans *Proc. CVPRW*, pages 554–561, 2013.

[50] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E HINTON.
« Imagenet classification with deep convolutional neural networks ».
Dans *Advances in neural information processing systems*, pages 1097–1105, 2012.

[51] Yann LECUN, John S DENKER et Sara A SOLLA.
« Optimal brain damage ».
Dans *proc of NIPS*, pages 598–605, 1990.

[52] J. T. LEE et Y. CHUNG.
« Deep Learning-Based Vehicle Classification Using an Ensemble of Local Expert and Global Networks ».
Dans *Proc. CVPRW*, pages 47–52, 2017.

[53] Tsung-Yi LIN, Michael MAIRE, Serge BELONGIE, James HAYS, Pietro PERONA, Deva RAMANAN, Piotr DOLLÁR et C Lawrence ZITNICK.
« Microsoft coco : Common objects in context ».
Dans *Proc. ECCV*, pages 740–755, 2014.

[54] Chenxi LIU, Barret ZOPH, Jonathon SHLENS, Wei HUA, Li-Jia LI, Li FEI-FEI, Alan YUILLE, Jonathan HUANG et Kevin MURPHY.
« Progressive neural architecture search ».
arXiv preprint arXiv :1712.00559, 2017.

[55] H. LIU, S. CHEN et N. KUBOTA.
« Intelligent Video Systems and Analytics : A Survey ».
IEEE Trans. Ind. Info., 9(3):1222–1233, 2013.

[56] Wei LIU, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU et Alexander C BERG.
« Ssd : Single shot multibox detector ».
Dans *proc. of ECCV*, pages 21–37, 2016.

[57] Z. LIU, J. LI, Z. SHEN, G.HUANG, S. YAN et C.ZHANG.
« Learning Efficient Convolutional Networks through Network Slimming ».
Dans *proc of ICCV*, 2017.

[58] Christos LOUIZOS, Max WELLING et Diederik P. KINGMA.
« Learning Sparse Neural Networks through L_0 Regularization ».
Dans *proc. of ICLR*, 2018.

[59] David G LOWE.
« Object recognition from local scale-invariant features ».
Dans *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157, 1999.

[60] Wenhan LUO, Xiaowei ZHAO et Tae-Kyun KIM.
« Multiple object tracking : A review ».
arXiv preprint arXiv :1409.7618, 1, 2014.

[61] Z LUO, F B.CHARRON, C LEMAIRE, J KONRAD, S LI, A MISHRA, A ACHKAR, J EICHEL et P-M JODOIN.

« MIO-TCD : A new benchmark dataset for vehicle classification and localization ».
In press at IEEE Trans. on Img. Proc., 2018.

[62] Zhiming LUO, Frédéric BRANCHAUD-CHARRON, Carl LEMAIRE, Janusz KONRAD, Shaozi LI, Akshaya Kumar MISHRA, Andrew ACHKAR, Justin EICHEL et Pierre-Marc JODOIN.
« MIO-TCD : A New Benchmark Dataset for Vehicle Classification and Localization ».
IEEE Transactions on Image Processing, 27:5129–5141, 2018.

[63] Zhiming LUO, Justin EICHEL, Andrew ACHKAR, Carl LEMAIRE, Janusz KONRAD, Akshaya MISHRA, Shaozi LI, Frederic B-CHARRON et Pierre-Marc JODOIN.
« Traffic Surveillance Workshop and Challenge (CVPR 2017) », 2017.
http://tcd.miovision.com.

[64] Chris J MADDISON, Andriy MNIH et Yee Whye TEH.
« The concrete distribution : A continuous relaxation of discrete random variables ».
proc of ICLR, 2017.

[65] Kevin MIO.
« Traffic jams », 2015.
http://montrealgazette.com/news/local-news/the-high-cost-of-congestion-to-montreals-economy.

[66] Thomas M. MITCHELL.
Machine Learning.
McGraw-Hill, Inc., 1 édition, 1997.

[67] Dmitry MOLCHANOV, Arsenii ASHUKHA et Dmitry VETROV.
« Variational dropout sparsifies deep neural networks ».
proc of ICML, 2017.

[68] Alexander MORDVINTSEV, Nicola PEZZOTTI, Ludwig SCHUBERT et Chris OLAH.
« Differentiable Image Parameterizations ».
Distill, 2018.
https ://distill.pub/2018/differentiable-parameterizations.

[69] A. Krizhevsky I. Sutskever R. Salakhutdinov N. SRIVASTAVA, J. Hinton.
« Dropout : A Simple Way to Prevent Neural Networks from Overfitting ».
Journal of ML Research, 15:1929–1958, 2014.

[70] Eric NALISNICK et Padhraic SMYTH.
« Unifying the Dropout Family Through Structured Shrinkage Priors ».
arXiv preprint arXiv :1810.04045, 2018.

[71] Kirill NEKLYUDOV, Dmitry MOLCHANOV, Arsenii ASHUKHA et Dmitry P VETROV.
« Structured bayesian pruning via log-normal multiplicative noise ».
Dans *proc of NIPS*, 2017.

[72] Chris OLAH, Alexander MORDVINTSEV et Ludwig SCHUBERT.
« Feature Visualization ».
Distill, 2017.
https ://distill.pub/2017/feature-visualization.

[73] Chris OLAH, Arvind SATYANARAYAN, Ian JOHNSON, Shan CARTER, Ludwig SCHUBERT, Katherine YE et Alexander MORDVINTSEV.
« The Building Blocks of Interpretability ».
Distill, 2018.
https ://distill.pub/2018/building-blocks.

[74] W. PAN, H. DONG et Y. GUO.
« DropNeuron : Simplifying the Structure of Deep Neural Networks ».
Dans *arXiv preprint arXiv :1606.07326*, 2016.

[75] Adam PASZKE et OTHERS.
« PyTorch », 2016.
https://github.com/pytorch/pytorch.

[76] Juan-Manuel PÉREZ-RÚA, Moez BACCOUCHE et Stephane PATEUX.
« Efficient Progressive Neural Architecture Search ».
arXiv preprint arXiv :1808.00391, 2018.

[77] Hieu PHAM, Melody Y GUAN, Barret ZOPH, Quoc V LE et Jeff DEAN.
« Efficient Neural Architecture Search via Parameter Sharing ».
arXiv preprint arXiv :1802.03268, 2018.

[78] Antonio POLINO, Razvan PASCANU et Dan ALISTARH.
« Model compression via distillation and quantization ».
Dans *International Conference on Learning Representations*, 2018.

[79] J. REDMON et A. FARHADI.
« YOLO9000 : Better, Faster, Stronger ».
Dans *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017.

[80] Joseph REDMON, Santosh DIVVALA, Ross GIRSHICK et Ali FARHADI.
« You only look once : Unified, real-time object detection ».
Dans *Proc. CVPR*, pages 779–788, 2016.

[81] Jonathan D. REGEHR et Rob POAPST.
« Traffic Counts : A new guide for traffic monitoring practitioners and traffic data customers in Canada ».
Transportation Talk, 39(4):23–27, Hiver 2017-2018.
https://issuu.com/cite7/docs/tt39.4-winter201718/26.

[82] Shaoqing REN, Kaiming HE, Ross GIRSHICK et Jian SUN.
« Faster r-cnn : Towards real-time object detection with region proposal networks ».
Dans *Proc. NIPS*, pages 91–99, 2015.

[83] Olaf RONNEBERGER, Philipp FISCHER et Thomas BROX.
« U-net : Convolutional networks for biomedical image segmentation ».
Dans *proc of MICCAI*, pages 234–241, 2015.

[84] Frank ROSENBLATT.
« The perceptron : a probabilistic model for information storage and organization in the brain. ».
Psychological review, 65(6):386, 1958.

[85] Olga RUSSAKOVSKY, Jia DENG, Hao SU, Jonathan KRAUSE, Sanjeev SATHEESH, Sean MA, Zhiheng HUANG, Andrej KARPATHY, Aditya KHOSLA, Michael BERNSTEIN et OTHERS.
« Imagenet large scale visual recognition challenge ».
Int. J. Comput. Vision, 115(3):211–252, 2015.

[86] C. E. SHANNON.
« A mathematical theory of communication ».
The Bell System Technical Journal, 27(3):379–423, July 1948.

[87] Ali SHARIF RAZAVIAN, Hossein AZIZPOUR, Josephine SULLIVAN et Stefan CARLSSON.
« CNN features off-the-shelf : an astounding baseline for recognition ».
Dans *Proc. CVPRW*, pages 806–813, 2014.

[88] Zhiqiang SHEN, Zhuang LIU, Jianguo LI, Yu-Gang JIANG, Yurong CHEN et Xiangyang XUE.
« Dsod : Learning deeply supervised object detectors from scratch ».
Dans *Proc. ICCV*, 2017.

[89] Karen SIMONYAN et Andrew ZISSERMAN.
« Very deep convolutional networks for large-scale image recognition ».
arXiv preprint arXiv :1409.1556, 2014.

[90] Christian SZEGEDY, Vincent VANHOUCKE, Sergey IOFFE, Jon SHLENS et Zbigniew WOJNA.
« Rethinking the inception architecture for computer vision ».
Dans *Proc. of CVPR*, pages 2818–2826, 2016.

[91] Rajkumar THEAGARAJAN, Federico PALA et Bir BHANU.
« EDeN : Ensemble of Deep Networks for Vehicle Classification ».
Dans *Proc. CVPRW*, pages 906–913, 2017.

[92] Jasper RR UIJLINGS, Koen EA VAN DE SANDE, Theo GEVERS et Arnold WM SMEULDERS.
« Selective search for object recognition ».
Int. J. Comput. Vision, 104(2):154–171, 2013.

[93] Oriol VINYALS, Alexander TOSHEV, Samy BENGIO et Dumitru ERHAN.
« Show and tell : A neural image caption generator ».
Dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[94] T. VÉNIAT et L. DENOYER.
« Learning Time/Memory-Efficient Deep Architectures with Budgeted Super Net-

works ».
Dans *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3492–3500, June 2018.

[95] Tao WANG, Xuming HE, Songzhi SU et Yin GUAN.
« Efficient Scene Layout Aware Object Detection for Traffic Surveillance ».
Dans *Proc. CVPRW*, pages 53–60, 2017.

[96] B-F WU, C-C KAO, J-H JUANG et Y-S HUANG.
« A New Approach to Video-Based Traffic Surveillance Using Fuzzy Hybrid Information Inference Mechanism ».
IEEE Trans. Int. Trans. Sys., 14(1):485–491, 2013.

[97] W.WEN, C WU, Y.WANG, Y CHEN et H.LI.
« Learning structured sparsity in deep neural networks ».
Dans *proc of NIPS*, 2016.

[98] Linjie YANG, Ping LUO, Chen CHANGE LOY et Xiaoou TANG.
« A large-scale car dataset for fine-grained categorization and verification ».
Dans *Proc. CVPR*, pages 3973–3981, 2015.

[99] Jason YOSINSKI, Jeff CLUNE, Yoshua BENGIO et Hod LIPSON.
« How transferable are features in deep neural networks ? ».
Dans *Proc. NIPS*, pages 3320–3328, 2014.

[100] Sergey ZAGORUYKO et Nikos KOMODAKIS.
« Wide Residual Networks ».
Dans *proc. of BMVC*, 2016.

[101] T. ZHANG, S LIU, C. XU et H. LU.
« Mining Semantic Context Information for Intelligent Video Surveillance of Traffic Scenes ».
IEEE Trans. Ind. Info., 9(1):149–160, 2013.

[102] Zhe ZHU, Dun LIANG, Songhai ZHANG, Xiaolei HUANG, Baoli LI et Shimin HU.
« Traffic-sign detection and classification in the wild ».
Dans *Proc. CVPR*, pages 2110–2118, 2016.

[103] Barret ZOPH et Quoc V LE.
« Neural architecture search with reinforcement learning ».
proc of ICLR, 2017.